Sidekiq API

Documentation and Reference

libsidekiq v4.18.x

Contents

Chapter 1

libsidekiq - Sidekiq Library

Sidekiq is a software defined radio card in a MiniPCIe, M.2 (3042 and 2280), or VITA 57.1 FPGA Mezzanine Card (FMC) form factor (Sidekiq X2 and X4). Each utilizes an RFIC, which provides the complete RF front end & baseband analog & A/D and D/A converters. An on-board FPGA then provides timestamping/buffering, along with optional signal processing.

For the MiniPCIe and M.2 form factors, a single lane (x1) PCIe interface in the FPGA provides a transport path between the host system and Sidekiq, which is used for streaming data between the host and Sidekiq, as well as for command/control of Sidekiq through a register interface. A USB 2.0 high speed interface is also included in Sidekiq mPCIe and M.2-3042, which is used to provide a path for re-programming the FPGA bitstream. This USB interface can also be used by the host for streaming of data and command/control of the card for host systems that include a MiniPCIe or M.2-3042 card slot but only wire up the USB 2.0 pins. See the `Epiq Solutions Website` for more details.

The Sidekiq Z2 is offered in a MiniPCIe form factor but uses a USB 2.0 high speed interface as a transport between the host system and the Zynq 7010 FPGA. See `Sidekiq Z2` for more details.

The VITA 57.1 FMC form factor can be used in conjunction with compliant FPGA carrier boards to provide a user with access to IQ samples and command / control. See `Sidekiq X2` and `Sidekiq X4` for more details.

The Sidekiq NV100 is offered in an M.2-2280 form factor and uses a Gen2 x2 PCIe as a transport between the on-board Artix 7 FPGA and the host system. See `Sidekiq NV100` for more details.

The following list enumerates the features of Sidekiq (MiniPCIe card form factor):

- Flexible RF front end supports two operating modes:
  - Two phase coherent RF receivers (common LO)
  - One RF receiver + one RF transmitter (separate LOs)
- RF tuning range from 70 MHz to 6 GHz
- Up to 50 MHz RF bandwidth per channel (min sample rate: 233 Ksps, max sample rate: 61.44 Msps)
- Great dynamic range with 12-bit A/D and D/A converters
- PCIe Gen 1 x1 (2.5 GT/s) interface to host + USB 2.0 Hi-Speed interface
- Integrated FPGA for custom signal processing and PCIe data transport to host
- Integrated temperature sensor + accelerometer

The following list enumerates the features of Sidekiq M.2 (M.2-3042 card form factor):

- Flexible RF front end supports two operating modes:
  - Two RF receiver + two RF transmitter (2x2 MIMO)
  - One RF receiver + one RF transmitter (separate LOs)
- RF tuning range from 70 MHz to 6 GHz
- Up to 50 MHz RF bandwidth per channel (min sample rate: 233 Ksps, max sample rate: 61.44 Msps)
- Great dynamic range with 12-bit A/D and D/A converters
- PCIe Gen 2 x1 (5.0GT/s) interface to host + USB 2.0 Hi-Speed interface
- Integrated FPGA for custom signal processing and PCIe data transport to host
- Integrated temperature sensor + accelerometer

The following list enumerates the features of Sidekiq Stretch (M.2-2280 Key B+M card form factor):

- One RF receiver + one RF transmitter (separate LOs)
- RF tuning range from 70 MHz to 6 GHz
- Up to 50 MHz RF bandwidth per channel (min sample rate: 233 Ksps, max sample rate: 61.44 Msps)
- Great dynamic range with 12-bit A/D and D/A converters
- PCIe x2 (5.0GT/s) interface to host
- Integrated FPGA for custom signal processing and PCIe data transport to host
- Integrated temperature sensor + accelerometer
- Integrated GPSDO receiver with 1PPS
- Sub-octave Rx pre-select filtering with adjustable band-pass from 150MHz to 6GHz

The following list enumerates the features of Sidekiq Z2 (MiniPCIe card form factor):

- Wideband RF Transceiver (Analog Devices' AD9364)
  - 1Rx + 1Tx RF Transceiver
  - RF tuning range from 70 MHz to 6 GHz
  - Four band Rx pre-select filter bank
  - Up to 61.44 Msps sample rate
  - Great dynamic range with 12-bit A/D and D/A converters
  - 40 MHz TCVCXO ref clock with +/- 1 PPM stability
- Linux Computer (Xilinx Zynq XC7Z010-2I)
  - Dual-core ARM Cortex A9 CPU running Linux
  - 512 MB of DDR3L RAM
  - 32 MB of QSPI flash memory
  - Linux boot time <2 seconds

The following list enumerates the features of Sidekiq X2 (VITA 57.1 FMC HPC form factor):

- Two phase coherent RF receivers (common LO) + third independently tunable RF receiver
- Seven band RF pre-select filters on all three Rx antenna ports
- Two phase coherent RF transmitters (common LO)
- RF tuning range from 1 MHz to 6 GHz
- Up to 100 MHz RF bandwidth per channel (max sample rate: 122.88 Msps)
- Exceptional dynamic range with 16-bit A/D converters, 14-bit D/A converters
- Integrated temperature sensor
- 10MHz + PPS sync inputs

The following list enumerates the features of Sidekiq X4 (VITA 57.1 FMC HPC form factor):

- Four RF receivers (phase coherent or **independently tunable**)
- Seven band-pass RF filters on each RF receiver
- Four RF transmitters (**phase coherent** or two phase coherent pairs)
- RF tuning range from 1 MHz to 6 GHz
- Up to 200 MHz RF bandwidth per channel (max sample rate: 245.76 Msps)
- Exceptional dynamic range with 16-bit A/D converters, 14-bit D/A converters
- Integrated temperature sensor
- 10MHz + PPS sync inputs

The following list enumerates the features of Matchstiq Z3u:

- Wideband RF Transceiver (Analog Devices' AD9364)
  - 2-channel phase coherent Rx, or 1 Tx + 1 Rx
  - RF tuning range from 70 MHz to 6 GHz
  - Up to 61.44 Msps sample rate
  - Great dynamic range with 12-bit A/D and D/A converters
  - 40 MHz TCVCXO ref clock with +/- 1 PPM stability
  - Integrated temperature sensor + 3-axis gyroscope + 3-axis accelerometer
  - Integrated GPSDO receiver with 1PPS
  - Sub-octave Rx pre-select filtering with adjustable band-pass from 150MHz to 6GHz
- Linux Computer (Xilinx Zynq Ultrascale+ XCZU3EG)
  - Quad-core ARM Cortex A53 CPU running Linux
  - 2 GB of LPDDR4 RAM
  - 128 MB of QSPI flash memory
  - 128 GB eMMC + microSD card slot
  - USB 3.0 OTG via USB-C

The following list enumerates the features of Sidekiq NV100:

- Wideband RF Transceiver (Analog Devices' ADRV9004)
  - Antenna Port 1: U.FL coaxial connector supporting Tx or Rx
  - Antenna Port 2: U.FL coaxial connector supporting either Tx or Rx
  - RF tuning range from 30 MHz to 6 GHz (RF access to 10 MHz)
  - Up to 40 MHz RF channel bandwidth
  - Up to 61.44 Msps sample rate
  - Exceptional RF fidelity and instantaneous dynamic range with 16-bit A/D and D/A converters
  - 40 MHz TCVCXO ref clock with +/- 1 PPM stability
  - Integrated temperature sensor + 3-axis gyroscope + 3-axis accelerometer
  - Integrated GPSDO receiver with 1PPS
  - Sub-octave Rx pre-select filtering from 400 MHz to 6 GHz

Documentation for the primary Sidekiq API exists in these files:

- sidekiq_api.h
- sidekiq_types.h
- sidekiq_params.h

Documentation for the custom transport developers, the Sidekiq Transport API, exists in these files:

- sidekiq_xport_api.h
- sidekiq_xport_types.h

Chapter 2

Deprecated List

Member RFIC_CONTROL_OUTPUT_MODE_GAIN_BITS

since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration.

Member RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA1

since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration to present A1 gain in the metadata.

Member RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA2

since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration to present A2 gain in the metadata.

Member skiq_hardware_vers_string (skiq_hw_vers_t hardware_vers)

Member skiq_hw_vers_t

Member SKIQ_MAX_LO_FREQ

To determine the max LO frequency use skiq_read_rx_LO_freq_range() or skiq_read_max_rx_LO_freq().

Member SKIQ_MAX_RX_GAIN

To determine the maximum Rx gain, use skiq_read_rx_gain_index_range().

Member SKIQ_MAX_SAMPLE_RATE

To determine the maximum sample rate for the specific hardware / radio configuration, refer to skiq_read_parameters.

Member SKIQ_MAX_TX_ATTENUATION

Use skiq_read_parameters() and the corresponding skiq_param_t struct to determine the attenuation range.

Member SKIQ_MIN_LO_FREQ

To determine the min LO frequency use skiq_read_rx_LO_freq_range() or skiq_read_min_rx_LO_freq().

Member SKIQ_MIN_RX_GAIN

To determine the minimum Rx gain, use skiq_read_rx_gain_index_range().

Member SKIQ_MIN_SAMPLE_RATE

To determine the minimum sample rate for the specific hardware / radio configuration, refer to skiq_read_parameters.

Member skiq_product_t

Member skiq_product_vers_string (skiq_product_t product_vers)

Member skiq_read_fpga_version (uint8_t card, uint32_t *p_git_hash, uint32_t *p_build_date, uint8_t *p_major, uint8_t *p_minor, skiq_fpga_tx_fifo_size_t *p_tx_fifo_size)

Use skiq_read_fpga_semantic_version() and skiq_read_fpga_tx_fifo_size() instead of skiq_read_fpga_version()

Member skiq_read_hw_version (uint8_t card, skiq_hw_vers_t *p_hw_version)

Member skiq_read_max_sample_rate (uint8_t card, uint32_t *p_max_sample_rate)

This function has been deprecated and may not return the correct maximum sample rate for all handles, this has been replaced with skiq_read_parameters.

Member skiq_read_min_sample_rate (uint8_t card, uint32_t *p_min_sample_rate)

This function has been deprecated and may not return the correct minimum sample rate for all handles, this has been replaced with skiq_read_parameters.

Member skiq_read_product_version (uint8_t card, skiq_product_t *p_product)

Member skiq_rf_port_config_tdd

use skiq_rf_port_config_trx

Member SKIQ_RX_META_HDL_BITS

Use skiq_rx_block_t::hdl instead of this definition

Member SKIQ_RX_META_OVERLOAD_BIT

Use skiq_rx_block_t::overload instead of this definition

Member SKIQ_RX_META_RFIC_CTRL_BITS

Use skiq_rx_block_t::rfic_control instead of this definition

Member SKIQ_RX_META_RFIC_CTRL_OFFSET

Use skiq_rx_block_t::rfic_control instead of this definition

Member SKIQ_RX_NUM_PACKETS_IN_RING_BUFFER

As of libsidekiq v4.13, this value is no longer guaranteed to be accurate as the value can change based upon the configuration of the PCI DMA Driver kernel module.

Member SKIQ_RX_SYS_META_WORD_OFFSET

Use skiq_rx_block_t::hdl, skiq_rx_block_t::overload, and skiq_rx_block_t::rfic_control instead of this definition

Member SKIQ_RX_USER_META_WORD_OFFSET

Use skiq_rx_block_t::user_meta instead of this definition

Member SKIQ_SYS_TIMESTAMP_FREQ

All platforms should use the skiq_read_sys_timestamp_freq() API instead

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Library Functions and Definitions

These functions and definitions are related to interacting with the library configuration unrelated to the Sidekiq SDR.

Functions

- EPIQ_API int32_t skiq_read_libsidekiq_version (uint8_t *p_major, uint8_t *p_minor, uint8_t *p_patch, const char **p_label)
- EPIQ_API void skiq_register_critical_error_callback (void(*critical_handler)(int32_t status, void *p_-user_data), void *p_user_data)
- EPIQ_API void skiq_register_logging (void(*log_msg)(int32_t priority, const char *message))
- EPIQ_API int32_t skiq_set_exit_handler_state (bool enabled)

  Set the state of the exit handler.

5.1.1 Detailed Description

These functions and definitions are related to interacting with the library configuration unrelated to the Sidekiq SDR.

5.1.2 Function Documentation

5.1.2.1 EPIQ_API int32_t skiq_read_libsidekiq_version (uint8_t * *p_major*, uint8_t * *p_minor*, uint8_t * *p_patch*, const char ** *p_label*)

The skiq_read_libsidekiq_version() function is responsible for returning the major/minor/patch/label revision numbers for the version of libsidekiq used by the application. The label revision will be a qualitative description of the revision rather than defining the API revision level.

Since

Function signature modified in API **v4.0.0** to add pointer to a revision label.

Parameters

out	*p_major*	a pointer to where the major rev # should be written
out	*p_minor*	a pointer to where the minor rev # should be written
out	*p_patch*	a pointer to where the patch rev # should be written
out	*p_label*	a pointer which will be set to point to a NULL-terminated string, which is possibly the empty string ""

Returns

int32_t status where 0=success, anything else is an error

5.1.2.2 EPIQ_API void skiq_register_critical_error_callback (void(∗)(int32_t status, void ∗p_user_data) *critical_handler,* void ∗ *p_user_data*)

The skiq_register_critical_error_callback() function allows a custom handler to be registered in the case of critical errors. If a critical error occurs and a callback function is registered, then the critical_handler will be called. If no handler is registered, then exit() is called. Continued use of libsidekiq after a critical error has occurred will result in undefined behavior.

See Also

skiq_register_logging

Parameters

in	*critical_handler*	function pointer to handler to call in the case of a critical error. If no handler is registered, exit() will be called.
in	*p_user_data*	a pointer to user data to be provided as an argument to the critical_-handler function when called. This can safely be set to NULL if not needed. However, this will cause the argument of the critical handler to also be set to NULL.

Returns

int32_t status where 0=success, anything else is an error

5.1.2.3 EPIQ_API void skiq_register_logging (void(∗)(int32_t priority, const char ∗message) *log_msg*)

The skiq_register_logging() function allows a custom logging handler to be registered. The priority (as by the SKIQ_LOG_∗ definitions) and the logging message are provided to the function. If no callback is registered, the logging messages are displayed in the console as well as syslog. If it is desired to completely disable any output of the library NULL can be registered for the logging function, in which case no logging will occur.

See Also

skiq_register_critical_error_callback

Parameters

in	*log_msg*	function pointer to handler to call when logging a message

Returns

int32_t status where 0=success, anything else is an error

5.1.2.4 EPIQ_API int32_t skiq_set_exit_handler_state (bool *enabled*)

Set the state of the exit handler.

Parameters

in	*enabled*	if false, disable the libsidekiq exit handler, else enable it.

By default, libsidekiq registers a handler function that is called when the running program is exited; this exit handler attempts to clean up after the library and free allocated resources. If this behavior is not desired for some reason, this function may be called with `state` set to false to bypass registering the exit handler.

Since

Function added in API **v4.14.0**

Note

The exit handler is installed after cards are initialized (using functions like skiq_init() or skiq_enable_-cards()), so this function must be called before card initialization.
The exit handler is not called if the host application crashes (for example, due to a segmentation fault). libsidekiq applications should still call skiq_exit() when access to the radios is no longer needed; the exit handler is installed as a safety measure to ensure proper cleanup.

Returns

0 on success

5.2 Card Functions and Definitions

These functions and definitions are related to initializing and configuring the digital (non-RF) related functionality of the Sidekiq SDR.

Classes

- struct skiq_part_info_t
  Sidekiq Part Information.

Macros

- #define SKIQ_SERIAL_NUM_STRLEN (6)
  Number of bytes contained in the serial number (including '\0')
- #define SKIQ_PART_NUM_STRLEN (7)
  Number of bytes contained in the part number (including '\0')
- #define SKIQ_REVISION_STRLEN (3)
  Number of bytes contained in the revision (including '\0')
- #define SKIQ_VARIANT_STRLEN (3)
  Number of bytes contained in the variant (including '\0')
- #define SKIQ_MAX_NUM_FILTERS (20)
  Maximum number of filters available for a handle.

Enumerations

- enum skiq_iq_order_t { skiq_iq_order_qi =0, skiq_iq_order_iq }
  An interface is configured to transmit or receive complex I/Q samples. By default samples are received/transmitted as I/Q pairs with 'Q' sample occurring first, followed by the 'I' sample. Ordering can be configured by the user at run-time before an Rx interface is started.
- enum skiq_filt_t {
  skiq_filt_invalid = -1, skiq_filt_0_to_3000_MHz =0, skiq_filt_3000_to_6000_MHz, skiq_filt_0_to_440-MHz,
  skiq_filt_440_to_6000MHz, skiq_filt_440_to_580MHz, skiq_filt_580_to_810MHz, skiq_filt_810_to_-1170MHz,
  skiq_filt_1170_to_1695MHz, skiq_filt_1695_to_2540MHz, skiq_filt_2540_to_3840MHz, skiq_filt_-3840_to_6000MHz,
  skiq_filt_0_to_300MHz, skiq_filt_300_to_6000MHz, skiq_filt_50_to_435MHz, skiq_filt_435_to_910M-Hz,
  skiq_filt_910_to_1950MHz, skiq_filt_1950_to_6000MHz, skiq_filt_0_to_6000MHz, skiq_filt_390_to_-620MHz,
  skiq_filt_540_to_850MHz, skiq_filt_770_to_1210MHz, skiq_filt_1130_to_1760MHz, skiq_filt_1680_to-_2580MHz,
  skiq_filt_2500_to_3880MHz, skiq_filt_3800_to_6000MHz, skiq_filt_47_to_135MHz, skiq_filt_135_to_-145MHz,
  skiq_filt_145_to_150MHz, skiq_filt_150_to_162MHz, skiq_filt_162_to_175MHz, skiq_filt_175_to_190-MHz,
  skiq_filt_190_to_212MHz, skiq_filt_212_to_230MHz, skiq_filt_230_to_280MHz, skiq_filt_280_to_366-MHz,
  skiq_filt_366_to_475MHz, skiq_filt_475_to_625MHz, skiq_filt_625_to_800MHz, skiq_filt_800_to_-

1175MHz,
skiq_filt_1175_to_1500MHz, skiq_filt_1500_to_2100MHz, skiq_filt_2100_to_2775MHz, skiq_filt_2775_to_3360MHz,
skiq_filt_3360_to_4600MHz, skiq_filt_4600_to_6000MHz, skiq_filt_30_to_450MHz, skiq_filt_450_to_600MHz,
skiq_filt_600_to_800MHz, skiq_filt_800_to_1200MHz, skiq_filt_1200_to_1700MHz, skiq_filt_1700_to_2700MHz,
skiq_filt_2700_to_3600MHz, skiq_filt_3600_to_6000MHz, skiq_filt_max }

  Each RF path in Sidekiq has integrated filter options that can be software-controlled. By default, the filter is automatically selected based on the requested LO frequency. The skiq_filt_t enum is used to specify a filter selection. Note: not all filter options are available for hardware variants. Available filter variants can be queried with skiq_read_rx_filters_avail.
- enum skiq_part_t {
  skiq_mpcie =0, skiq_m2, skiq_x2, skiq_z2,
  skiq_x4, skiq_m2_2280, skiq_z2p, skiq_z3u,
  skiq_nv100, skiq_part_invalid }

  Sidekiq Part.
- enum skiq_ref_clock_select_t {
  skiq_ref_clock_internal =0, skiq_ref_clock_external, skiq_ref_clock_host, skiq_ref_clock_carrier_edge,
  skiq_ref_clock_invalid }

  Reference clock setting.

Functions

- EPIQ_API int32_t skiq_get_cards (skiq_xport_type_t xport_type, uint8_t *p_num_cards, uint8_t *p_cards)
- EPIQ_API int32_t skiq_read_serial_string (uint8_t card, char **pp_serial_num)
- EPIQ_API int32_t skiq_get_card_from_serial_string (char *p_serial_num, uint8_t *p_card)
- EPIQ_API int32_t skiq_init (skiq_xport_type_t type, skiq_xport_init_level_t level, uint8_t *p_card_nums, uint8_t num_cards)
- EPIQ_API int32_t skiq_enable_cards (const uint8_t cards[ ], uint8_t num_cards, skiq_xport_init_level_t level)
- EPIQ_API int32_t skiq_enable_cards_by_serial_str (const char **pp_serial_nums, uint8_t num_cards, skiq_xport_init_level_t level, uint8_t *p_card_nums)
- EPIQ_API int32_t skiq_init_by_serial_str (skiq_xport_type_t type, skiq_xport_init_level_t level, char **pp_serial_nums, uint8_t num_cards, uint8_t *p_card_nums)
- EPIQ_API int32_t skiq_init_without_cards (void)
- EPIQ_API int32_t skiq_read_parameters (uint8_t card, skiq_param_t *p_param)
- EPIQ_API int32_t skiq_is_xport_avail (uint8_t card, skiq_xport_type_t type)
- EPIQ_API int32_t skiq_is_card_avail (uint8_t card, pid_t *p_card_owner)
- EPIQ_API int32_t skiq_exit (void)
- EPIQ_API int32_t skiq_disable_cards (const uint8_t cards[ ], uint8_t num_cards)
- EPIQ_API int32_t skiq_read_temp (uint8_t card, int8_t *p_temp_in_deg_C)
- EPIQ_API int32_t skiq_read_fw_version (uint8_t card, uint8_t *p_major, uint8_t *p_minor)
- EPIQ_API const char * skiq_part_string (skiq_part_t part)
- EPIQ_API int32_t skiq_read_part_info (uint8_t card, char *p_part_number, char *p_revision, char *p_variant)
- EPIQ_API int32_t skiq_read_ref_clock_select (uint8_t card, skiq_ref_clock_select_t *p_ref_clk)
- EPIQ_API int32_t skiq_read_ext_ref_clock_freq (uint8_t card, uint32_t *p_freq)
- EPIQ_API int32_t skiq_read_rx_filters_avail (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t *p_filters, uint8_t *p_num_filters)

- EPIQ_API int32_t skiq_read_tx_filters_avail (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t ∗p_filters, uint8_t ∗p_num_filters)
- EPIQ_API int32_t skiq_read_filter_range (skiq_filt_t filter, uint64_t ∗p_start_freq, uint64_t ∗p_end_freq)
- EPIQ_API int32_t skiq_read_usb_enumeration_delay (uint8_t card, uint16_t ∗p_delay_ms)
- EPIQ_API int32_t skiq_read_calibration_date (uint8_t card, uint16_t ∗p_last_cal_year, uint8_t ∗p_last_cal_week, uint8_t ∗p_cal_interval)
- EPIQ_API int32_t skiq_write_ref_clock_select (uint8_t card, skiq_ref_clock_select_t ref_clock_source)

  This function allows the user to switch between different reference clock sources. This change is run-time only and is not written to the card nor permanent.
- EPIQ_API int32_t skiq_write_ext_ref_clock_freq (uint8_t card, uint32_t ext_freq)

  This function allows the user to switch between different external reference clock frequencies. This change is run-time only and is not written to the card nor permanent. This will automatically update the reference clock selection to an external reference clock source. When changing the frequency, a supported external reference clock frequency must be used per the card specification.

Variables

- EPIQ_API const char ∗ SKIQ_FILT_STRINGS [skiq_filt_max]

  String representation of skiq_filt_t enumeration.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_MPCIE_001

  String representation of the Sidekiq mPCIe 001 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_MPCIE_002

  String representation of the Sidekiq mPCIe 002 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_M2

  String representation of the Sidekiq m.2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_X2

  String representation of the Sidekiq X2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z2

  String representation of the Sidekiq Z2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_X4

  String representation of the Sidekiq X4 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_M2_2280

  String representation of the Sidekiq M.2 2280 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z2P

  String representation of the Sidekiq Z2+ part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z3U

  String representation of the Sidekiq Z3u part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_NV100

  String representation of the Sidekiq NV100 part.

5.2.1 Detailed Description

These functions and definitions are related to initializing and configuring the digital (non-RF) related functionality of the Sidekiq SDR.

5.2.2 Macro Definition Documentation

5.2.2.1 #define SKIQ_SERIAL_NUM_STRLEN (6)

Number of bytes contained in the serial number (including '\0')

Definition at line 77 of file sidekiq_types.h.

5.2.2.2 #define SKIQ_PART_NUM_STRLEN (7)

Number of bytes contained in the part number (including '\0')

Definition at line 79 of file sidekiq_types.h.

5.2.2.3 #define SKIQ_REVISION_STRLEN (3)

Number of bytes contained in the revision (including '\0')

Definition at line 81 of file sidekiq_types.h.

5.2.2.4 #define SKIQ_VARIANT_STRLEN (3)

Number of bytes contained in the variant (including '\0')

Definition at line 83 of file sidekiq_types.h.

5.2.2.5 #define SKIQ_MAX_NUM_FILTERS (20)

Maximum number of filters available for a handle.

See Also

skiq_read_rx_filters_avail

Definition at line 89 of file sidekiq_types.h.

5.2.3 Enumeration Type Documentation

5.2.3.1 enum skiq_iq_order_t

An interface is configured to transmit or receive complex I/Q samples. By default samples are received/transmitted as I/Q pairs with 'Q' sample occurring first, followed by the 'I' sample. Ordering can be configured by the user at run-time before an Rx interface is started.

```
              skiq_iq_order_qi: (default)                skiq_iq_order_iq:
            -15--------------------------0-      -15--------------------------0-
            |          12-bit Q0_A1        |      |          12-bit I0_A1        |
index 0     | (sign extended to 16 bits)  |      | (sign extended to 16 bits)  |
            -------------------------------      -------------------------------
            |          12-bit I0_A1        |      |          12-bit Q0_A1        |
index 1     | (sign extended to 16 bits)  |      | (sign extended to 16 bits)  |
            -------------------------------      -------------------------------
```

```
          |         12-bit Q1_A1         |      |         12-bit I1_A1         |
index 2   | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
          ------------------------------        ------------------------------
          |         12-bit I1_A1         |      |         12-bit Q1_A1         |
index 3   | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
          ------------------------------        ------------------------------
          |             ...              |      |             ...              |
          ------------------------------        ------------------------------
          |             ...              |      |             ...              |
          -15--------------------------0-       -15--------------------------0-
```

See Also

skiq_read_iq_order_mode
skiq_write_iq_order_mode

Enumerator

skiq_iq_order_qi

skiq_iq_order_iq

Definition at line 301 of file sidekiq_types.h.

5.2.3.2 enum skiq_filt_t

Each RF path in Sidekiq has integrated filter options that can be software-controlled. By default, the filter is automatically selected based on the requested LO frequency. The skiq_filt_t enum is used to specify a filter selection. Note: not all filter options are available for hardware variants. Available filter variants can be queried with skiq_read_rx_filters_avail.

See Also

skiq_read_rx_filters_avail
skiq_read_filter_range
skiq_read_rx_preselect_filter_path
skiq_read_tx_filters_avail

Enumerator

skiq_filt_invalid

skiq_filt_0_to_3000_MHz

skiq_filt_3000_to_6000_MHz

skiq_filt_0_to_440MHz

skiq_filt_440_to_6000MHz

skiq_filt_440_to_580MHz

skiq_filt_580_to_810MHz

skiq_filt_810_to_1170MHz

skiq_filt_1170_to_1695MHz

skiq_filt_1695_to_2540MHz

skiq_filt_2540_to_3840MHz

skiq_filt_3840_to_6000MHz
skiq_filt_0_to_300MHz
skiq_filt_300_to_6000MHz
skiq_filt_50_to_435MHz
skiq_filt_435_to_910MHz
skiq_filt_910_to_1950MHz
skiq_filt_1950_to_6000MHz
skiq_filt_0_to_6000MHz
skiq_filt_390_to_620MHz
skiq_filt_540_to_850MHz
skiq_filt_770_to_1210MHz
skiq_filt_1130_to_1760MHz
skiq_filt_1680_to_2580MHz
skiq_filt_2500_to_3880MHz
skiq_filt_3800_to_6000MHz
skiq_filt_47_to_135MHz
skiq_filt_135_to_145MHz
skiq_filt_145_to_150MHz
skiq_filt_150_to_162MHz
skiq_filt_162_to_175MHz
skiq_filt_175_to_190MHz
skiq_filt_190_to_212MHz
skiq_filt_212_to_230MHz
skiq_filt_230_to_280MHz
skiq_filt_280_to_366MHz
skiq_filt_366_to_475MHz
skiq_filt_475_to_625MHz
skiq_filt_625_to_800MHz
skiq_filt_800_to_1175MHz
skiq_filt_1175_to_1500MHz
skiq_filt_1500_to_2100MHz
skiq_filt_2100_to_2775MHz
skiq_filt_2775_to_3360MHz
skiq_filt_3360_to_4600MHz
skiq_filt_4600_to_6000MHz
skiq_filt_30_to_450MHz
skiq_filt_450_to_600MHz
skiq_filt_600_to_800MHz
skiq_filt_800_to_1200MHz
skiq_filt_1200_to_1700MHz
skiq_filt_1700_to_2700MHz
skiq_filt_2700_to_3600MHz
skiq_filt_3600_to_6000MHz
skiq_filt_max

Definition at line 411 of file sidekiq_types.h.

5.2.3.3 enum skiq_part_t

Sidekiq Part.

See Also

skiq_part_string
skiq_hardware_vers_string
skiq_product_vers_string

Enumerator

skiq_mpcie

skiq_m2

skiq_x2

skiq_z2

skiq_x4

skiq_m2_2280

skiq_z2p

skiq_z3u

skiq_nv100

skiq_part_invalid

Definition at line 587 of file sidekiq_types.h.

5.2.3.4 enum skiq_ref_clock_select_t

Reference clock setting.

Warning

This setting is **NOT** software configurable for Sidekiq mPCIe skiq_hw_vers_mpcie_b

See Also

skiq_read_ext_ref_clock_freq
skiq_read_ref_clock_select

Enumerator

skiq_ref_clock_internal use the default internal reference clock

skiq_ref_clock_external an external, user-accessible reference clock

skiq_ref_clock_host reference clock originated from host

Since

enum added in **v4.5.0**

skiq_ref_clock_carrier_edge reference clock originated from carrier

skiq_ref_clock_invalid

Definition at line 704 of file sidekiq_types.h.

5.2.4 Function Documentation

5.2.4.1 EPIQ_API int32_t skiq_get_cards (skiq_xport_type_t *xport_type*, uint8_t ∗ *p_num_cards*, uint8_t ∗ *p_cards*)

The skiq_get_cards() function is responsible for generating a list of valid Sidekiq card indices for the transport specified. Return of the card does not mean that it is available for use by the application. To check card availability, refer to skiq_is_card_avail().

Since

Function added in API **v4.0.0**

Note

Can be called before skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str()

See Also

skiq_init
skiq_init_by_serial_str
skiq_exit

Parameters

in	*xport_type*	[skiq_xport_type_t] transport type to detect card
out	*p_num_cards*	pointer to where to store the number of cards
out	*p_cards*	pointer to where to store the card indices of the Sidekiqs available. There should be room to store at least SKIQ_MAX_NUM_CARDS at this location.

Returns

int32_t status where 0=success, anything else is an error

5.2.4.2 EPIQ_API int32_t skiq_read_serial_string (uint8_t *card*, char ∗∗ *pp_serial_num*)

The skiq_read_serial_string() function is responsible for returning the serial number of the Sidekiq.

Note

Memory used for holding the string representation of the serial number is managed internally by libsidekiq and does not need to be managed in any manner by the end user (i.e. no need to free memory).

See Also

skiq_get_card_from_serial_string

Parameters

in	*card*	card index of the Sidekiq of interest
out	*pp_serial_num*	a pointer to hold the serial number

Returns

int32_t: status where 0=success, anything else is an error

5.2.4.3 EPIQ_API int32_t skiq_get_card_from_serial_string (char ∗ *p_serial_num,* uint8_t ∗ *p_card*)

The skiq_get_card_from_serial_string() function is responsible for obtaining the Sidekiq card index for the specified serial number.

See Also

skiq_read_serial_string

Parameters

in	*p_serial_num*	serial number of Sidekiq card
out	*p_card*	pointer to where to store the corresponding card index of the specified Sidekiq

Returns

int32_t status where 0=success, anything else is an error

5.2.4.4 EPIQ_API int32_t skiq_init (skiq_xport_type_t *type,* skiq_xport_init_level_t *level,* uint8_t ∗ *p_card_nums,* uint8_t *num_cards*)

The skiq_init() function is responsible for performing all initialization tasks for the sidekiq platform.

Since

Function signature modified in API **v4.0.0**

See Also

skiq_init_without_cards
skiq_init_by_serial_str
skiq_exit
skiq_enable_cards
skiq_enable_cards_by_serial_str
skiq_disable_cards

Parameters

in	*type*	[skiq_xport_type_t] the transport type that is required: • skiq_xport_type_auto - automatically detect and use available transport • skiq_xport_type_pcie - communicate with Sidekiq over PCIe. If USB is available it will also be used for certain functionality. • skiq_xport_type_usb - communicate with Sidekiq entirely over USB. A USB FPGA bitstream must be utilized if initializing at skiq_xport_init_level_full. • skiq_xport_type_custom - communicate with Sidekiq using the registered transport implementation provided by a call to skiq_register_custom_transport(). If USB is available, it will also be used for certain functionality.
in	*level*	[skiq_xport_init_level_t] the transport functionality level of initialization that is required: • skiq_xport_init_level_basic - minimal initialization necessary to bring up the requested transport interface for FPGA / RFIC register reads/writes, and initialize the mutexes that serializes access to libsidekiq • skiq_xport_init_level_full - Same as skiq_xport_init_level_basic and perform the complete bring up of all hardware (most applications concerned with sending/receiving RF will use this)
in	*p_card_nums*	pointer to the list of Sidekiq card indices to be initialized
in	*num_cards*	number of Sidekiq cards to initialize

Attention

As of libsidekiq v4.8.0, the *type* parameter is ignored as the transport type is automatically set to skiq_xport_type_auto, which will select the correct transport for the specified card(s).
skiq_init() and skiq_init_by_serial_str() should only be called when starting an application or after skiq_exit() has been called; these functions are not designed to be called multiple times to initialize individual cards.

Returns

int32_t status where 0=success, anything else is an error

Return values

-EEXIST	libsidekiq has already been initialized in this application without skiq_exit() being called
-E2BIG	if the number of cards requested exceeds the maximum (SKIQ_MAX_NUM_CARDS)

-EINVAL	if one of the specified card indices is out of range or refers to a non-existent card

5.2.4.5 EPIQ_API int32_t skiq_enable_cards (const uint8_t *cards[ ]*, uint8_t *num_cards*, skiq_xport_init_level_t *level*)

The skiq_enable_cards() function is responsible for performing all initialization tasks for the specified Sidekiq cards.

Attention

The Sidekiq library must have been previously initialized with skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str(). The transport type is automatically selected based on availability.

Since

Function added in API **v4.8.0**

See Also

skiq_init
skiq_init_without_cards
skiq_init_by_serial_str
skiq_exit
skiq_enable_cards_by_serial_str
skiq_disable_cards

Parameters

in	*cards*	array of Sidekiq card indices to be initialized
in	*num_cards*	number of Sidekiq cards to initialize
in	*level*	[skiq_xport_init_level_t] the transport functionality level of initialization that is required: • skiq_xport_init_level_basic - minimal initialization necessary to bring up the requested transport interface for FPGA / RFIC register reads/writes, and initialize the mutexes that serializes access to libsidekiq • skiq_xport_init_level_full - Same as skiq_xport_init_level_basic and perform the complete bring up of all hardware (most applications concerned with sending/receiving RF will use this)

Returns

int32_t status where 0=success, anything else is an error

Return values

-EPERM	if libsidekiq has not been initialized yet (through skiq_init(), skiq_init_-without_cards(), or skiq_init_by_serial_str())
-EINVAL	if one of the specified card indices is out of range or refers to a non-existent card
-E2BIG	if the number of cards specified exceeds the maximum (SKIQ_MAX_NUM_-CARDS)
-EBUSY	if one or more of the specified cards is already in use (either by the current process or another)

5.2.4.6 EPIQ_API int32_t skiq_enable_cards_by_serial_str (const char ** *pp_serial_nums,* uint8_t *num_cards,* skiq_xport_init_level_t *level,* uint8_t * *p_card_nums*)

The skiq_enable_cards_by_serial_str() function is responsible for performing all initialization tasks for the specified Sidekiq cards.

Attention

The Sidekiq library must have been previously initialized with skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str(). The transport type is automatically selected based on availability.

Since

Function added in API **v4.9.0**

See Also

skiq_init
skiq_init_without_cards
skiq_init_by_serial_str
skiq_exit
skiq_enable_cards
skiq_disable_cards

Parameters

`in`	*pp_serial_nums*	pointer to the list of Sidekiq serial number strings to initialize
`in`	*num_cards*	number of Sidekiq cards to initialize
`in`	*level*	the transport functionality level of initialization that is required: • skiq_xport_init_level_basic - minimal initialization necessary to bring up the requested transport interface for FPGA / RFIC register reads/writes, and initialize the mutexes that serializes access to libsidekiq • skiq_xport_init_level_full - Same as skiq_xport_init_level_basic and perform the complete bring up of all hardware (most applications concerned with sending/receiving RF will use this)
`out`	*p_card_nums*	pointer to the list of Sidekiq card indices corresponding with serial strings provided; this list should be able to hold at least SKIQ_MAX_NUM_CARDS entries

Returns

int32_t status where 0=success, anything else is an error

Return values

-EPERM	if libsidekiq has not been initialized yet (through skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str())
-E2BIG	if the number of cards specified exceeds the maximum (SKIQ_MAX_NUM_CARDS)
-ENXIO	if one of the specified serial numbers cannot be obtained

5.2.4.7 EPIQ_API int32_t skiq_init_by_serial_str (skiq_xport_type_t *type*, skiq_xport_init_level_t *level*, char ** *pp_serial_nums*, uint8_t *num_cards*, uint8_t * *p_card_nums*)

The skiq_init_by_serial_str() function is identical to skiq_init() except a list of serial numbers can be requested instead of card indices.

Since

Function added in API **v4.0.0**

See Also

skiq_init
skiq_exit

Parameters

in	*type*	[skiq_xport_type_t] the transport type that is required: • skiq_xport_type_auto - automatically detect and use available transport • skiq_xport_type_pcie - communicate with Sidekiq over PCIe. If USB is available it will also be used for certain functionality. • skiq_xport_type_usb - communicate with Sidekiq entirely over USB. A USB FPGA bitstream must be utilized if initializing at skiq_xport_init_level_full. • skiq_xport_type_custom - communicate with Sidekiq using the registered transport implementation provided by a call to skiq_register_custom_transport(). If USB is available, it will also be used for certain functionality.

in	*level*	the transport functionality level of initialization that is required: • skiq_xport_init_level_basic - minimal initialization necessary to bring up the requested transport interface for FPGA / RFIC register reads/writes, and initialize the mutexes that serializes access to libsidekiq • skiq_xport_init_level_full - Same as skiq_xport_init_level_basic and perform the complete bring up of all hardware (most applications concerned with sending/receiving RF will use this)
in	*pp_serial_nums*	pointer to the list of Sidekiq serial number strings to initialize
in	*num_cards*	number of Sidekiq cards to initialize
out	*p_card_nums*	pointer to the list of Sidekiq card indices corresponding with serial strings provided; this list should be able to hold at least SKIQ_MAX_NUM_CARDS entries

Attention

As of libsidekiq v4.8.0, the *type* parameter is ignored as the transport type is automatically set to skiq_xport_type_auto, which will select the correct transport for the specified card(s).
skiq_init(), skiq_init_without_cards(), and skiq_init_by_serial_str() should only be called when starting an application or after skiq_exit() has been called; these functions are not designed to be called multiple times to initialize individual cards.

Returns

int32_t status where 0=success, anything else is an error

Return values

-EEXIST	libsidekiq has already been initialized in this application without skiq_exit() being called
-E2BIG	if the number of cards requested exceeds the maximum (SKIQ_MAX_NUM_CARDS)
-ENXIO	if one of the specified serial numbers cannot be found

5.2.4.8 EPIQ_API int32_t skiq_init_without_cards (void)

The skiq_init_without_cards() function initializes the library (like skiq_init()) without having to specify any cards. This is useful when using cards dynamically via the skiq_enable_cards() / skiq_disable_cards() functions.

Attention

skiq_init(), skiq_init_without_cards(), and skiq_init_by_serial_str() should only be called when starting an application or after skiq_exit() has been called; these functions are not designed to be called multiple times.

Since

Function added in API **v4.13.0**

See Also

skiq_init
skiq_init_by_serial_str
skiq_enable_cards
skiq_disable_cards
skiq_exit

Returns

int32_t status where 0 = success, anything else is an error

Return values

-EEXIST	libsidekiq has already been initialized in this application without skiq_exit() being called

5.2.4.9 EPIQ_API int32_t skiq_read_parameters (uint8_t *card*, skiq_param_t ∗ *p_param*)

The skiq_read_parameters() function is used for populating the skiq_param_t struct for a given card. This structure can be queried for various values relating to the card. For further information regarding that structure, reference the documentation provided in sidekiq_params.h.

Note

The initialization level influences what can be populated in the structure. This is fully documented in skiq_params.h.

Since

Function added in API **v4.4.0**

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_param*	[skiq_param_t] pointer to structure to be populated.

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_param` is NULL
-EPROTO	if an internal error is detected

5.2.4.10 EPIQ_API int32_t skiq_is_xport_avail (uint8_t *card*, skiq_xport_type_t *type*)

The skiq_is_xport_avail() function is responsible for determining if the requested transport type is available for the card index specified.

Since

Function added in API **v4.0.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*type*	transport type to check for card specified

Returns

int32_t status where 0=success, anything else is an error

5.2.4.11 EPIQ_API int32_t skiq_is_card_avail (uint8_t *card*, pid_t * *p_card_owner*)

The skiq_is_card_avail() function is responsible for determining if the requested card is currently available and free for use. If the card is already locked, the process ID of the current card owner is provided.

Note

This only reflects the instantaneous availability of the Sidekiq card and does not reserve any resources for future use.
If a card is locked by another thread within the current process, the process ID (PID) returned in p_card_owner can be the PID of the current process.

Since

Function added in API **v4.0.0**

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_card_owner*	is a pointer where the process ID of the current card owner is provided (only if the card is already locked). May be NULL if the caller does not require the information; if not NULL, this value is set if the function returns 0 or EBUSY.

Returns

int32_t status code

Return values

-ERANGE	if the specified card index exceeds the maximum (SKIQ_MAX_NUM_CARDS)
-ENODEV	if a card was not detected at the specified card index
0	if the card is available
EBUSY	if the specified card is not available (already in use)
non-zero	Unspecified error occurred

5.2.4.12 EPIQ_API int32_t skiq_exit (void)

The skiq_exit() function is responsible for performing all shutdown tasks for libsidekiq. It should be called once when the associated application is closing.

See Also

skiq_init
skiq_init_by_serial_str

Returns

int32_t status where 0=success, anything else is an error

5.2.4.13 EPIQ_API int32_t skiq_disable_cards (const uint8_t *cards[ ]*, uint8_t *num_cards*)

The skiq_disable_cards() function is responsible for performing all shutdown tasks for the specified Sidekiq card(s). This does not perform the various shutdown tasks for all of libsidekiq, only for the card(s) specified.

Since

Function added in API **v4.8.0**

Attention

The Sidekiq library must have been previously initialized with:

- skiq_init(),
- skiq_init_without_cards(),
- or skiq_init_by_serial_str() and the specified card(s) must have been initialized with either:
- skiq_init(),
- skiq_init_by_serial_str(),
- skiq_enable_cards(), or
- skiq_enable_cards_by_serial_str().

This function does not automatically release all libsidekiq resources if all cards are disabled; if libsidekiq is no longer needed, skiq_exit() must be called to perform a clean shutdown of the library.

See Also

skiq_init
skiq_init_by_serial_str
skiq_enable_cards
skiq_enable_cards_by_serial_str

Parameters

in	*cards*	array of Sidekiq cards to be disabled
in	*num_cards*	number of Sidekiq cards to disabled

Returns

int32_t status where 0=success, anything else is an error

Return values

-EPERM	if libsidekiq has not been initialized yet (through skiq_init(), skiq_init_-without_cards(), or skiq_init_by_serial_str())
-E2BIG	if the number of cards requested exceeds the maximum (SKIQ_MAX_NUM_-CARDS)
-EINVAL	if one of the specified card indices is out of range or refers to a non-existent card

5.2.4.14 EPIQ_API int32_t skiq_read_temp (uint8_t *card,* int8_t ∗ *p_temp_in_deg_C*)

The skiq_read_temp() function is responsible for reading and providing the current temperature of the unit (in degrees Celsius).

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_temp_in_deg-_C*	a pointer to where the current temp should be written

Returns

0 on success, else a negative errno value

Return values

-EAGAIN	Temperature sensor measurement is temporarily not available, try again later
-ENODEV	Temperature sensor not available in present skiq_xport_init_level_t, try skiq-_xport_init_level_full
-EINVAL	No supported sensors found
-EIO	I/O communication error occurred during measurement
-ENOTSUP	No sensors for associated Sidekiq product

5.2.4.15 EPIQ_API int32_t skiq_read_fw_version (uint8_t *card,* uint8_t ∗ *p_major,* uint8_t ∗ *p_minor*)

The skiq_read_fw_version() function is responsible for returning the major/minor revision numbers for the microcontroller firmware within the Sidekiq unit

Note

This is currently only supported if the USB interface has been initialized.

Attention

This function is valid only for mPCIe and M.2 and will otherwise return an error.

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_major*	a pointer to where the major rev # should be written
out	*p_minor*	a pointer to where the minor rev # should be written

Returns

int32_t status where 0=success, anything else is an error

5.2.4.16 EPIQ_API const char∗ skiq_part_string (skiq_part_t *part*)

The skiq_part_string() function returns a string representation of the passed in part value.

Since

Function added in API **v4.4.0**

Parameters

in	*part*	[skiq_part_t] Sidekiq part value

Returns

const char∗ string representing the Sidekiq part

5.2.4.17 EPIQ_API int32_t skiq_read_part_info (uint8_t *card,* char ∗ *p_part_number,* char ∗ *p_revision,* char ∗ *p_variant*)

The skiq_part_info() function returns strings representing the various components of a part.

Since

Function added in API **v4.2.0**

Parameters

in	*card*	card index of Sidekiq of interest
out	*p_part_number*	pointer to where to store the part number (ex: "020201") Must be able to contain SKIQ_PART_NUM_STRLEN # of bytes.
out	*p_revision*	pointer to where to store the revision. (ex: "B0") Must be able to contain SKIQ_REVISION_STRLEN # of bytes.
out	*p_variant*	pointer to where to store the variant. (ex: "01") Must be able to contain SKIQ_VARIANT_STRLEN # of bytes.

Returns

int32_t status where 0=success, anything else is an error

5.2.4.18 EPIQ_API int32_t skiq_read_ref_clock_select (uint8_t *card*, skiq_ref_clock_select_t * *p_ref_clk*)

The skiq_read_ref_clock_select() function is responsible for reading the reference clock configuration.

Attention

this is not supported on rev B mPCIe

See Also

skiq_read_ext_ref_clock_freq
skiq_write_ref_clock_select

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_ref_clk*	[skiq_ref_clock_select_t] pointer to where to store the reference clock setting

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.2.4.19 EPIQ_API int32_t skiq_read_ext_ref_clock_freq (uint8_t *card*, uint32_t * *p_freq*)

The skiq_read_ext_ref_clock_freq() function is responsible for reading the external reference clock's configured frequency.

Note

The default value is 40MHz if not configured.
This function is only supported for mPCIe and M.2 Sidekiq variants.

Since

Function added in API **v4.2.0**

See Also

skiq_read_ref_clock_select

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_freq*	pointer to where to store the external clock's frequency

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.2.4.20 EPIQ_API int32_t skiq_read_rx_filters_avail (uint8_t *card,* skiq_rx_hdl_t *hdl,* skiq_filt_t ∗ *p_filters,* uint8_t ∗ *p_num_filters*)

The skiq_read_rx_filters_avail() function allows an application to obtain the preselect filters available for the specified card and handle.

Note

By default, when the LO frequency of the handle is adjusted, the filter encompassing the configured LO frequency is automatically configured.

Warning

There will never be more than skiq_filt_invalid filters returned and p_filters should be sized such that it can hold that many filter values.

Since

Function added in API **v4.2.0**

See Also

skiq_read_filter_range
skiq_read_rx_preselect_filter_path
skiq_read_tx_filters_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] RX handle of the filter availability in question
out	*p_filters*	[:skiq_filt_t] pointer to list of filters available
out	*p_num_filters*	pointer to where to store the number of filters

Returns

int32_t status where 0=success, anything else is an error

5.2.4.21 EPIQ_API int32_t skiq_read_tx_filters_avail (uint8_t *card,* skiq_tx_hdl_t *hdl,* skiq_filt_t ∗ *p_filters,* uint8_t ∗ *p_num_filters*)

The skiq_read_tx_filters_avail() function allows an application to obtain the preselect filters available for the specified card and handle.

Note

by default, when the LO frequency of the handle is adjusted, the filter encompassing the configured LO frequency is automatically configured.

Warning

There will never be more than skiq_filt_invalid filters returned and p_filters should be sized such that it can hold that many filter values.

Since

Function added in API **v4.2.0**

See Also

skiq_read_filter_range
skiq_read_tx_filter_path
skiq_read_rx_filters_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] TX handle of the filter availability in question
out	*p_filters*	[skiq_filt_t] pointer to list of filters available
out	*p_num_filters*	pointer to where to store the number of filters

Returns

int32_t status where 0=success, anything else is an error

5.2.4.22 EPIQ_API int32_t skiq_read_filter_range (skiq_filt_t *filter*, uint64_t ∗ *p_start_freq*, uint64_t ∗ *p_end_freq*)

The skiq_read_filter_range() function provides a mechanism to determine the frequency range covered by the specified filter.

Since

Function added in API **v4.2.0**

See Also

skiq_read_tx_filters_avail
skiq_read_rx_filters_avail

Parameters

in	*filter*	[skiq_filt_t] filter of interest
out	*p_start_freq*	pointer to where to store the start frequency covered by the filter
out	*p_end_freq*	pointer to where to store the end frequency covered by the filter

Returns

int32_t status where 0=success, anything else is an error

5.2.4.23 EPIQ_API int32_t skiq_read_usb_enumeration_delay (uint8_t *card*, uint16_t ∗ *p_delay_ms*)

The skiq_read_usb_enumeration_delay() function reads the number of milliseconds that the Sidekiq should delay USB enumeration, if supported.

Warning

This function will return an error if called on a unit that does not have an FX2 placed on it.

Since

Function added in API **v4.2.0**, requires firmware **v2.7** or later

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_delay_ms*	pointer to take total enumeration delay in milliseconds

Returns

int32_t status where 0=success, anything else is an error

5.2.4.24 EPIQ_API int32_t skiq_read_calibration_date (uint8_t *card*, uint16_t ∗ *p_last_cal_year*, uint8_t ∗ *p_last_cal_week*, uint8_t ∗ *p_cal_interval*)

The skiq_read_calibration_date() function reads details on when calibration was last performed. Additionally, a recommended date to perform the next calibration is provided.

Since

Function added in API **v4.7.0**

See Also

skiq_read_part_info
skiq_read_parameters

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_last_cal_year*	pointer to where to store the year when calibration was last performed.
out	*p_last_cal_week*	pointer to where to store the week number when the calibration was last performed. The week number with the calibration year provides a full representation of when the calibration was performed.
out	*p_cal_interval*	pointer to where to store the interval (in years) of how often calibration should be performed. The year of the last calibration (adjusted by this interval) along with the week of the last calibration provides a recommendation for when the next calibration should be performed.

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENOENT	Calibration date information cannot be located

5.2.4.25 EPIQ_API int32_t skiq_write_ref_clock_select (uint8_t *card*, skiq_ref_clock_select_t *ref_clock_source*)

This function allows the user to switch between different reference clock sources. This change is run-time only and is not written to the card nor permanent.

Note

For non-volatile storage of reference clock configuration see ref_clock test app.

Warning

Sidekiq M.2 (skiq_m2) and Sidekiq mPCIe (skiq_mpcie) runtime reference clock source configuration is not supported.
Programming the reference clock dynamically using this function will initiate a full RF initialization process. The user should either call this function prior to RF configuration or reconfigure RF parameters after invoking this function, otherwise the user specified configuration will be lost.

See Also

skiq_read_ext_ref_clock_freq
skiq_read_ref_clock_select

Since

Function added in API **v4.14.0**

Parameters

`in`	*card*	requested Sidekiq card ID
`in`	*ref_clock_source*	[skiq_ref_clock_select_t] requested reference clock source to switch card to

Returns

0 on success, else a negative errno value

Return values

-EINVAL	if the requested reference select is invalid

-ENOTSUP	if the requested card is not supported
-ERANGE	if the requested card is not within the valid range of all cards
-ENODEV	if the requested card is not activated

5.2.4.26 EPIQ_API int32_t skiq_write_ext_ref_clock_freq (uint8_t *card*, uint32_t *ext_freq*)

This function allows the user to switch between different external reference clock frequencies. This change is run-time only and is not written to the card nor permanent. This will automatically update the reference clock selection to an external reference clock source. When changing the frequency, a supported external reference clock frequency must be used per the card specification.

Note

For non-volatile storage of external clock frequency configuration see ref_clock test app.
Runtime reference clock frequency switching is only supported on Sidekiq Stretch (skiq_m2_2280) and Sidekiq NV100 (skiq_nv100) (as of libsidekiq v4.17.0).

Warning

Switching the reference clock frequency here will stop receiving and transmitting.
Programming the reference clock dynamically using this function will initiate a full RF initialization process. The user should either call this function prior to RF configuration or reconfigure RF parameters after invoking this function, otherwise the user specified configuration will be lost.

See Also

skiq_read_ext_ref_clock_freq
skiq_read_ref_clock_select

Since

Function added in API **v4.17.0**

Parameters

`in`	*card*	requested Sidekiq card ID
`in`	*ext_freq*	requested external reference clock frequency to switch to (10MHz, or 40-MHz on both Stretch and NV100 and Stretch also supports 30.72MHz)

Returns

0 on success, else a negative errno value

Return values

-EINVAL	if the requested frequency is invalid
-ENOTSUP	if the requested card is not supported

-ERANGE	if the requested card is not within the valid range of all cards
-ENODEV	if the requested card is not activated

5.2.5 Variable Documentation

5.2.5.1 EPIQ_API const char* SKIQ_FILT_STRINGS[skiq_filt_max]

String representation of skiq_filt_t enumeration.

See Also

skiq_filt_t

Definition at line 484 of file sidekiq_types.h.

5.2.5.2 EPIQ_API const char* SKIQ_PART_NUM_STRING_MPCIE_001

String representation of the Sidekiq mPCIe 001 part.

Definition at line 498 of file sidekiq_types.h.

5.2.5.3 EPIQ_API const char* SKIQ_PART_NUM_STRING_MPCIE_002

String representation of the Sidekiq mPCIe 002 part.

Definition at line 500 of file sidekiq_types.h.

5.2.5.4 EPIQ_API const char* SKIQ_PART_NUM_STRING_M2

String representation of the Sidekiq m.2 part.

Definition at line 502 of file sidekiq_types.h.

5.2.5.5 EPIQ_API const char* SKIQ_PART_NUM_STRING_X2

String representation of the Sidekiq X2 part.

Definition at line 504 of file sidekiq_types.h.

5.2.5.6 EPIQ_API const char* SKIQ_PART_NUM_STRING_Z2

String representation of the Sidekiq Z2 part.

Definition at line 506 of file sidekiq_types.h.

5.2.5.7 EPIQ_API const char* SKIQ_PART_NUM_STRING_X4

String representation of the Sidekiq X4 part.

Definition at line 508 of file sidekiq_types.h.

5.2.5.8 EPIQ_API const char* SKIQ_PART_NUM_STRING_M2_2280

String representation of the Sidekiq M.2 2280 part.

Definition at line 510 of file sidekiq_types.h.

5.2.5.9 EPIQ_API const char* SKIQ_PART_NUM_STRING_Z2P

String representation of the Sidekiq Z2+ part.

Definition at line 512 of file sidekiq_types.h.

5.2.5.10 EPIQ_API const char* SKIQ_PART_NUM_STRING_Z3U

String representation of the Sidekiq Z3u part.

Definition at line 514 of file sidekiq_types.h.

5.2.5.11 EPIQ_API const char* SKIQ_PART_NUM_STRING_NV100

String representation of the Sidekiq NV100 part.

Definition at line 516 of file sidekiq_types.h.

5.3 Timestamp Functions

These functions are related to configuring and querying the System and RF timestamps of the Sidekiq SDR.

Functions

- EPIQ_API int32_t skiq_read_curr_rx_timestamp (uint8_t card, skiq_rx_hdl_t hdl, uint64_t ∗p_timestamp)
- EPIQ_API int32_t skiq_read_curr_tx_timestamp (uint8_t card, skiq_tx_hdl_t hdl, uint64_t ∗p_timestamp)
- EPIQ_API int32_t skiq_read_curr_sys_timestamp (uint8_t card, uint64_t ∗p_timestamp)
- EPIQ_API int32_t skiq_reset_timestamps (uint8_t card)
- EPIQ_API int32_t skiq_update_timestamps (uint8_t card, uint64_t new_timestamp)
- EPIQ_API int32_t skiq_read_sys_timestamp_freq (uint8_t card, uint64_t ∗p_sys_timestamp_freq)

5.3.1 Detailed Description

These functions are related to configuring and querying the System and RF timestamps of the Sidekiq SDR.

5.3.2 Function Documentation

5.3.2.1 EPIQ_API int32_t skiq_read_curr_rx_timestamp (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t ∗ *p_timestamp*)

The skiq_read_curr_rx_timestamp() function is responsible for retrieving a current snapshot of the Rx timestamp counter (i.e., free-running counter) of the specified interface handle. This timestamp is maintained by the FPGA and is shared across each RFIC regardless of the Rx or Tx interface.

Note

by the time the timestamp has been returned back to software, it will already be in the past, but this is still useful to determine if a specific timestamp has occurred already or not.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_read_curr_tx_timestamp
skiq_read_curr_sys_timestamp
skiq_reset_timestamps
skiq_update_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the interface for which the current timestamp is being read
out	*p_timestamp*	a pointer to where the 64-bit timestamp value should be written

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EDOM	if the requested handle is not available or out of range for the Sidekiq platform
-EFAULT	if `p_timestamp` is NULL
-EBADMSG	if an error occurred transacting with FPGA registers

5.3.2.2 EPIQ_API int32_t skiq_read_curr_tx_timestamp (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t * *p_timestamp*)

The skiq_read_curr_tx_timestamp() function is responsible for retrieving the currently set value for the timestamp (i.e., free-running counter) of the specified interface handle. This timestamp is maintained by the FPGA and is shared across each RFIC regardless of the Rx or Tx interface.

Note

by the time the timestamp has been returned back to software, it will already be in the past, but this is still useful to determine if a specific timestamp has occurred already or not.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_read_curr_rx_timestamp
skiq_read_curr_sys_timestamp
skiq_reset_timestamps
skiq_update_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the interface for which the current timestamp is being read

out	*p_timestamp*	a pointer to where the 64-bit timestamp value should be written

Returns

int32_t status where 0=success, anything else is an error

5.3.2.3 EPIQ_API int32_t skiq_read_curr_sys_timestamp (uint8_t *card*, uint64_t * *p_timestamp*)

The skiq_read_curr_sys_timestamp() function is responsible for retrieving the currently set value for the system timestamp. The system timestamp increments at the SKIQ_SYS_TIMESTAMP_FREQ rate. This timestamp is maintained by the FPGA and increments independent of the sample rate.

Note

by the time the timestamp has been returned back to software, it will already be in the past, but this is still useful to determine if a specific timestamp has occurred already or not.

See Also

skiq_read_curr_rx_timestamp
skiq_read_curr_tx_timestamp
skiq_reset_timestamps
skiq_update_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_timestamp*	a pointer to where the 64-bit timestamp value should be written

Returns

int32_t status where 0=success, anything else is an error

5.3.2.4 EPIQ_API int32_t skiq_reset_timestamps (uint8_t *card*)

The skiq_reset_timestamp() function is responsible for resetting the timestamps (Rx/Tx and system) back to 0.

See Also

skiq_read_curr_rx_timestamp
skiq_read_curr_tx_timestamp
skiq_read_curr_sys_timestamp
skiq_update_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest

Returns

int32_t status where 0=success, anything else is an error

5.3.2.5 EPIQ_API int32_t skiq_update_timestamps (uint8_t *card*, uint64_t *new_timestamp*)

The skiq_update_timestamps() function is responsible for updating the both the RF and system timestamps to the value specified.

See Also

skiq_read_curr_rx_timestamp
skiq_read_curr_tx_timestamp
skiq_read_curr_sys_timestamp
skiq_reset_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest
in	*new_timestamp*	value to set both the RF and system timestamps to

Returns

int32_t status where 0=success, anything else is an error

5.3.2.6 EPIQ_API int32_t skiq_read_sys_timestamp_freq (uint8_t *card*, uint64_t * *p_sys_timestamp_freq*)

The skiq_read_sys_timestamp_freq() function reads the system timestamp frequency (in Hz). This API replaces usage of SKIQ_SYS_TIMESTAMP_FREQ. This frequency represents the frequency at which the System Timestamp increments.

Attention

On the Sidekiq X2 platform, this frequency value may change when the receive or transmit sample rate changes.

Since

Function added in API **v4.2.0**

Parameters

in	*card*	card index of the Sidekiq of interest

out	*p_sys_timestamp_freq*	pointer to where to store the system timestamp frequency

Returns

int32_t status where 0=success, anything else is an error

Return values

0	successful query of the system timestamp frequency
-EINVAL	specified card index is out of range
-EINVAL	reference to p_sys_timestamp_freq is NULL
-ENODEV	specified card index has not been initialized

5.4 RFIC Functions and Definitions

These functions and definitions are related to configuring and exercising functionality of the RFIC on the Sidekiq SDR. These functions may also be related to receive and transmit capabilities, but are grouped here because they deal directly with the RFIC configuration.

Functions

- EPIQ_API int32_t skiq_prog_rfic_from_file (FILE *fp, uint8_t card)
- EPIQ_API int32_t skiq_read_rfic_reg (uint8_t card, uint16_t addr, uint8_t *p_data)
- EPIQ_API int32_t skiq_write_rfic_reg (uint8_t card, uint16_t addr, uint8_t data)
- EPIQ_API int32_t skiq_read_rfic_tx_fir_config (uint8_t card, uint8_t *p_num_taps, uint8_t *p_fir_interpolation)
- EPIQ_API int32_t skiq_read_rfic_tx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_write_rfic_tx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_read_rfic_rx_fir_config (uint8_t card, uint8_t *p_num_taps, uint8_t *p_fir_decimation)
- EPIQ_API int32_t skiq_read_rfic_rx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_write_rfic_rx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_read_rfic_control_output_rx_gain_config (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *p_mode, uint8_t *p_ena)
- EPIQ_API int32_t skiq_write_rfic_control_output_config (uint8_t card, uint8_t mode, uint8_t ena)
- EPIQ_API int32_t skiq_read_rfic_control_output_config (uint8_t card, uint8_t *p_mode, uint8_t *p_ena)
- EPIQ_API int32_t skiq_enable_rfic_control_output_rx_gain (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_write_rx_rfic_pin_ctrl_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rfic_pin_mode_t mode)

  skiq_write_rx_rfic_pin_ctrl_mode selects the source of RFIC Rx enable on supported RFICs. This signal disables or enables the receiver signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.
- EPIQ_API int32_t skiq_write_tx_rfic_pin_ctrl_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_rfic_pin_mode_t mode)

  skiq_write_tx_rfic_pin_ctrl_mode selects the source of RFIC Tx enable on supported RFICs. This signal disables or enables the transmitter signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.
- EPIQ_API int32_t skiq_read_rx_rfic_pin_ctrl_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rfic_pin_mode_t *p_mode)

  This function reads the source of control used to enable/disable RFIC Rx.
- EPIQ_API int32_t skiq_read_tx_rfic_pin_ctrl_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_rfic_pin_mode_t *p_mode)

  This function reads the source of control used to enable/disable RFIC Tx.

5.4.1 Detailed Description

These functions and definitions are related to configuring and exercising functionality of the RFIC on the Sidekiq SDR. These functions may also be related to receive and transmit capabilities, but are grouped here because they deal directly with the RFIC configuration.

5.4.2 Function Documentation

5.4.2.1 EPIQ_API int32_t skiq_prog_rfic_from_file (FILE * *fp*, uint8_t *card*)

The skiq_prog_rfic_from_file() function is responsible for pushing down a configuration file to the RFIC to reconfigure it. This allows libsidekiq-based apps to reconfigure the RFIC from a config file at run-time if needed.

Note

As of **v3.5.0**, programming the RFIC with a default configuration is part of skiq_init(), skiq_init_by_serial_str(), or skiq_enable_cards().

Parameters

in	*fp*	pointer to the already opened file to load to the RFIC
in	*card*	card index of the Sidekiq of interest

Returns

int32_t status where 0=success, anything else is an error

5.4.2.2 EPIQ_API int32_t skiq_read_rfic_reg (uint8_t *card*, uint16_t *addr*, uint8_t * *p_data*)

The skiq_read_rfic_reg() function reads the value of the RFIC register specified.

See Also

skiq_write_rfic_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*addr*	RFIC register address to read
out	*p_data*	pointer to where to store the value read

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.4.2.3 EPIQ_API int32_t skiq_write_rfic_reg (uint8_t *card*, uint16_t *addr*, uint8_t *data*)

The skiq_write_rfic_reg() function writes the data specified to the RFIC register provided.

Attention

writing directly to RFIC registers is not recommended. Modifying register settings may result in incorrect or unexpected behavior.

See Also

skiq_read_rfic_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*addr*	RFIC register address to write to
in	*data*	value to actually write to the register

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.4.2.4 EPIQ_API int32_t skiq_read_rfic_tx_fir_config (uint8_t *card*, uint8_t * *p_num_taps*, uint8_t * *p_fir_interpolation*)

The skiq_read_rfic_tx_fir_config() function provides access to the current number of Tx FIR taps as well as the Tx FIR interpolation.

Warning

any modification of the sample rate and/or channel bandwidth may result in a change of the number of taps and/or the interpolation factor.

See Also

skiq_read_rfic_tx_fir_coeffs
skiq_write_rfic_tx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_num_taps*	pointer to where to store the number of taps
out	*p_fir_interpolation*	pointer to where to store the interpolation factor of the Tx FIR

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.5 EPIQ_API int32_t skiq_read_rfic_tx_fir_coeffs (uint8_t *card*, int16_t * *p_coeffs*)

The skiq_read_rfic_tx_fir_coeffs() function provides access to the current Tx FIR coefficients programmed. To determine the number of taps and the interpolation factor of the FIR, use skiq_read_rfic_tx_fir_config().

Warning

any modification of the sample rate and/or channel bandwidth will result in an update of the FIR configuration and coefficients.

See Also

skiq_read_rfic_tx_fir_config
skiq_write_rfic_tx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_coeffs*	pointer to where to store the FIR coefficients

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.6 EPIQ_API int32_t skiq_write_rfic_tx_fir_coeffs (uint8_t *card*, int16_t ∗ *p_coeffs*)

The skiq_write_rfic_tx_fir_coeffs() function allows the coefficients of the Tx FIR to be written. The number of taps and interpolation factor are determined by the sample rate and can be queried using skiq_read_rfic_tx_fir_config().

Note

Any modification of the Rx/Tx sample rate and/or channel bandwidth will result in a change of the coefficients programmed. If a custom setting is used, the Rx/Tx sample rate and bandwidth must be performed first (skiq_write_rx_sample_rate_and_bandwidth() and skiq_write_tx_sample_rate_and_bandwidth()) after which skiq_write_rfic_tx_fir_coeffs() may be called. Additionally, the analog filters will be configured based on the configured channel bandwidth. For any sample rate which results in a interpolation setting of 4 results in the automatic doubling of FIR coefficients. The skiq_read_rfic_tx_fir_coeffs() returns the actual coefficient values programmed.

Attention

Writing the FIR coefficients directly using this function is not recommended. Modifying the FIR coefficients may result in incorrect or unexpected behavior.

See Also

skiq_read_rfic_tx_fir_config
skiq_read_rfic_tx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_coeffs*	pointer to where the Tx FIR coefficients are located

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.7 EPIQ_API int32_t skiq_read_rfic_rx_fir_config (uint8_t *card*, uint8_t ∗ *p_num_taps*, uint8_t ∗ *p_fir_decimation*)

The skiq_read_rfic_rx_fir_config() function provides access to the current number of Rx FIR taps as well as the Rx FIR decimation.

Warning

any modification of the sample rate and/or channel bandwidth may result in a change of number of taps and/or the decimation factor.

See Also

skiq_read_rfic_rx_fir_coeffs
skiq_write_rfic_rx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_num_taps*	pointer to where to store the number of taps
out	*p_fir_-decimation*	pointer to where to store the FIR decimation factor

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.8 EPIQ_API int32_t skiq_read_rfic_rx_fir_coeffs (uint8_t *card,* int16_t * *p_coeffs*)

The skiq_read_rfic_rx_fir_coeffs() function provides access to the current Rx FIR coefficients programmed. To determine the number of taps and the decimation factor of the Rx FIR, use skiq_read_rfic_rx_fir_config().

Warning

any modification of the sample rate and/or channel bandwidth will result in of the FIR configuration and coefficients.

See Also

skiq_read_rfic_rx_fir_config
skiq_write_rfic_rx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_coeffs*	pointer to where to store the FIR coefficients

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.9 EPIQ_API int32_t skiq_write_rfic_rx_fir_coeffs (uint8_t *card,* int16_t * *p_coeffs*)

The skiq_write_rfic_rx_fir_coeffs() function allows the coefficients of the Rx FIR to be written. The number of taps and interpolation factor are determined by the sample rate and can be queried using skiq_read_rfic_-rx_fir_config().

Note

any modification of the Rx/Tx sample rate and/or channel bandwidth will result in a change of the coefficients programmed. If a custom setting is used, the Rx/Tx sample rate and bandwidth must be performed first (skiq_write_rx_sample_rate_and_bandwidth() and skiq_write_tx_sample_rate_and_bandwidth()) after which skiq_write_rfic_rx_fir_coeffs() may be called. Additionally, the analog filters will be configured based on the configured channel bandwidth.

Attention

Writing the FIR coefficients directly using this function is not recommended. Modifying the FIR coefficients may result in incorrect or unexpected behavior.

See Also

skiq_read_rfic_rx_fir_config
skiq_read_rfic_rx_fir_coeffs

Parameters

in	*card*	card index of the Sidekiq of interest
in	*p_coeffs*	pointer to where the Rx FIR coefficients are located

Returns

int32_t status of the operation (0=success, anything else is an error)

5.4.2.10 EPIQ_API int32_t skiq_read_rfic_control_output_rx_gain_config (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t * *p_mode*, uint8_t * *p_ena*)

The skiq_read_rfic_control_output_rx_gain_config() function provides the mode and enable settings to configure the control output to present the gain of the handle specified.

See Also

skiq_write_rfic_control_output_config
skiq_read_rfic_control_output_config
skiq_rx_block_t::rfic_control

Since

Function added in **v4.9.0**, requires FPGA **v3.11.0** or later for Sidekiq X2 and X4

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] RX handle of the gain setting to present in control output

out	*p_mode*	pointer to where to store the control output mode setting
out	*p_ena*	pointer to where to store the control output enable setting

Returns

int32_t status where 0=success, anything else is an error

5.4.2.11 EPIQ_API int32_t skiq_write_rfic_control_output_config (uint8_t *card*, uint8_t *mode*, uint8_t *ena*)

The skiq_write_rfic_control_output_config() function allows the control output configuration of the RFIC to be configured. The control output readings are included within each receive packet's metadata (skiq_rx_block_t::rfic_control).

For details on the fields available for the control output, refer to the "Monitor Output" section of the appropriate reference manual.

- For Sidekiq mPCIe / m.2 / Z2, refer to p.73 of the `AD9361 Reference Manual UG-570`
- For Sidekiq X2, refer to Table 142 on p.192 of the AD9371 User Guide (UG-992)
- For Sidekiq X4, refer to Table 130 on p.214 of the ADRV9008-1/ADRV9008-2/ADRV9009 Hardware Reference Manual UG-1295

See Also

skiq_read_rfic_control_output_config
skiq_rx_block_t::rfic_control
skiq_read_rfic_control_output_rx_gain_config

Parameters

in	*card*	card index of the Sidekiq of interest
in	*mode*	control output mode
in	*ena*	control output enable

Returns

int32_t status where 0=success, anything else is an error

5.4.2.12 EPIQ_API int32_t skiq_read_rfic_control_output_config (uint8_t *card*, uint8_t * *p_mode*, uint8_t * *p_ena*)

The skiq_read_rfic_control_output_config() function allows the control output configuration of the RFIC to be read.

For details on the fields available for the control output, refer to the "Monitor Output" section of the appropriate reference manual.

For Sidekiq mPCIe / m.2 / Z2, refer to p.73 of the `AD9361 Reference Manual UG-570`:

- For Sidekiq X2, refer to Table 142 on p.192 of the AD9371 User Guide (UG-992):
- For Sidekiq X4, refer to Table 130 on p.214 of the ADRV9008-1/ADRV9008-2/ADRV9009 Hardware Reference Manual UG-1295:

See Also

skiq_write_rfic_control_output_config
skiq_read_rfic_control_output_rx_gain_config
skiq_rx_block_t::rfic_control

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_mode*	pointer to where to store the control output mode setting
out	*p_ena*	pointer to where to store the control output enable setting

Returns

int32_t status where 0=success, anything else is an error

5.4.2.13 EPIQ_API int32_t skiq_enable_rfic_control_output_rx_gain (uint8_t *card*, skiq_rx_hdl_t *hdl*)

The skiq_enable_rfic_control_output_rx_gain() function applies the RFIC mode and enable settings to configure the control output to represent the gain of the handle specified. This is equivalent to calling skiq_read_rfic_control_output_rx_gain_config() followed by skiq_write_rfic_control_output_config() with the appropriate mode and enable settings for the RX handle.

See Also

skiq_write_rfic_control_output_config
skiq_read_rfic_control_output_config
skiq_rx_block_t::rfic_control

Since

Function added in **v4.9.0**, requires FPGA **v3.11.0** or later for Sidekiq X2 and X

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] RX handle of the gain setting to present in control output

Returns

int32_t status where 0=success, anything else is an error

5.4.2.14 EPIQ_API int32_t skiq_write_rx_rfic_pin_ctrl_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rfic_pin_mode_t *mode*)

skiq_write_rx_rfic_pin_ctrl_mode selects the source of RFIC Rx enable on supported RFICs. This signal disables or enables the receiver signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.

Attention

Modifying RFIC pin control mode on `Sidekiq X4`Sidekiq X4 (skiq_x4) is supported starting in **v4.-14.0** while other Sidekiq products are not supported at this version. For details regarding GPIO pin mappings, please refer to the "FMC Pin Map" section of `Sidekiq X4 Hardware User's Manual`.

Since

Function added in API **v4.14.0**

See Also

skiq_read_rx_rfic_pin_ctrl_mode

Parameters

`in`	*card*	requested Sidekiq card ID
`in`	*hdl*	[skiq_rx_hdl_t] handle of the requested rx interface
`in`	*mode*	[skiq_rfic_pin_mode_t] desired mode

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-ENOTSUP	if the requested mode isn't supported for this card

5.4.2.15 EPIQ_API int32_t skiq_write_tx_rfic_pin_ctrl_mode (uint8_t *card,* skiq_tx_hdl_t *hdl,* skiq_rfic_pin_mode_t *mode*)

skiq_write_tx_rfic_pin_ctrl_mode selects the source of RFIC Tx enable on supported RFICs. This signal disables or enables the transmitter signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.

Attention

Modifying RFIC pin control mode on `Sidekiq X4`Sidekiq X4 (skiq_x4) is supported starting in **v4.-14.0** while other Sidekiq products are not supported at this version. For details regarding GPIO pin mappings, please refer to the "FMC Pin Map" section of `Sidekiq X4 Hardware User's Manual`.

Since

Function added in API **v4.14.0**

See Also

skiq_read_tx_rfic_pin_ctrl_mode

Parameters

in	*card*	requested Sidekiq card ID
in	*hdl*	[skiq_tx_hdl_t] handle of the requested Tx interface
in	*mode*	[skiq_rfic_pin_mode_t] desired mode

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-ENOTSUP	if the requested mode isn't supported for this card

5.4.2.16 EPIQ_API int32_t skiq_read_rx_rfic_pin_ctrl_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rfic_pin_mode_t ∗ *p_mode*)

This function reads the source of control used to enable/disable RFIC Rx.

Attention

Modifying RFIC pin control mode on `Sidekiq X4`Sidekiq X4 (skiq_x4) is supported starting in **v4.-14.0** while other Sidekiq products are not supported at this version. For details regarding GPIO pin mappings, please refer to the "FMC Pin Map" section of `Sidekiq X4 Hardware User's Manual`.

Since

Function added in API **v4.14.0**

See Also

skiq_write_rfic_pin_ctrl_mode

Parameters

in	*card*	requested Sidekiq card ID
in	*hdl*	[skiq_rx_hdl_t] handle of the requested Rx interface
out	*p_mode*	pointer to [skiq_rfic_pin_mode_t] configured mode

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range

-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.4.2.17 EPIQ_API int32_t skiq_read_tx_rfic_pin_ctrl_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_rfic_pin_mode_t * *p_mode*)

This function reads the source of control used to enable/disable RFIC Tx.

Attention

Modifying RFIC pin control mode on `Sidekiq X4`Sidekiq X4 (skiq_x4) is supported starting in **v4.-14.0** while other Sidekiq products are not supported at this version. For details regarding GPIO pin mappings, please refer to the "FMC Pin Map" section of `Sidekiq X4 Hardware User's Manual`.

Since

Function added in API **v4.14.0**

See Also

skiq_write_tx_rfic_pin_ctrl_mode

Parameters

`in`	*card*	requested Sidekiq card ID
`in`	*hdl*	[skiq_tx_hdl_t] handle of the requested Tx interface
`out`	*p_mode*	pointer to [skiq_rfic_pin_mode_t] configured mode

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.5 RF Port Functions and Definitions

These functions and definitions are related to configuring and exercising the capabilities of the physical RF connections and pathways of the Sidekiq SDR.

Enumerations

- enum skiq_rf_port_config_t { skiq_rf_port_config_fixed = 0, skiq_rf_port_config_tdd, skiq_rf_port_config_trx, skiq_rf_port_config_invalid }

  RF port configuration options of Sidekiq.
- enum skiq_rf_port_t {
  skiq_rf_port_unknown =-1, skiq_rf_port_J1 = 0, skiq_rf_port_J2, skiq_rf_port_J3,
  skiq_rf_port_J4, skiq_rf_port_J5, skiq_rf_port_J6, skiq_rf_port_J7,
  skiq_rf_port_J300, skiq_rf_port_Jxxx_RX1, skiq_rf_port_Jxxx_TX1RX2, skiq_rf_port_J8,
  skiq_rf_port_max }

  RF ports of Sidekiq.

Functions

- EPIQ_API const char ∗ skiq_rf_port_string (skiq_rf_port_t rf_port)
- EPIQ_API int32_t skiq_read_rf_port_config_avail (uint8_t card, bool ∗p_fixed, bool ∗p_trx)
- EPIQ_API int32_t skiq_read_rf_port_config (uint8_t card, skiq_rf_port_config_t ∗p_config)
- EPIQ_API int32_t skiq_write_rf_port_config (uint8_t card, skiq_rf_port_config_t config)
- EPIQ_API int32_t skiq_read_rf_port_operation (uint8_t card, bool ∗p_transmit)
- EPIQ_API int32_t skiq_write_rf_port_operation (uint8_t card, bool transmit)
- EPIQ_API int32_t skiq_read_rx_rf_ports_avail_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, uint8_t ∗p_num_fixed_rf_ports, skiq_rf_port_t ∗p_fixed_rf_port_list, uint8_t ∗p_num_trx_rf_ports, skiq_rf_port_t ∗p_trx_rf_port_list)
- EPIQ_API int32_t skiq_read_rx_rf_port_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t ∗p_rf_port)
- EPIQ_API int32_t skiq_write_rx_rf_port_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t rf_port)
- EPIQ_API int32_t skiq_read_tx_rf_ports_avail_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, uint8_t ∗p_num_fixed_rf_ports, skiq_rf_port_t ∗p_fixed_rf_port_list, uint8_t ∗p_num_trx_rf_ports, skiq_rf_port_t ∗p_trx_rf_port_list)
- EPIQ_API int32_t skiq_read_tx_rf_port_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, skiq_rf_port_t ∗p_rf_port)
- EPIQ_API int32_t skiq_write_tx_rf_port_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, skiq_rf_port_t rf_port)

Variables

- EPIQ_API const char ∗ SKIQ_RF_PORT_STRINGS [skiq_rf_port_max]

  String representation of skiq_rf_port_t enumeration.

5.5.1 Detailed Description

These functions and definitions are related to configuring and exercising the capabilities of the physical RF connections and pathways of the Sidekiq SDR.

5.5.2 Enumeration Type Documentation

5.5.2.1 enum skiq_rf_port_config_t

RF port configuration options of Sidekiq.

See Also

skiq_read_rf_port_config_avail
skiq_read_rf_port_config
skiq_write_rf_port_config
skiq_read_rf_port_operation
skiq_write_rf_port_operation

Enumerator

skiq_rf_port_config_fixed a single RF port can be used for either Rx OR Tx but not both

skiq_rf_port_config_tdd TDD: a single RF port can switch between Rx AND Tx, duplexed over time with the skiq_write_rf_port_operation() API

Deprecated use skiq_rf_port_config_trx

skiq_rf_port_config_trx TRx ports can be used for either receive or transmit, duplexed over time with the skiq_write_rf_port_operation() API

skiq_rf_port_config_invalid

Definition at line 768 of file sidekiq_types.h.

5.5.2.2 enum skiq_rf_port_t

RF ports of Sidekiq.

See Also

skiq_read_rx_rf_ports_avail_for_hdl
skiq_read_rx_rf_port_for_hdl
skiq_write_rx_rf_port_for_hdl
skiq_read_tx_rf_ports_avail_for_hdl
skiq_read_tx_rf_port_for_hdl
skiq_read_tx_rf_port_for_hdl

Enumerator

skiq_rf_port_unknown

skiq_rf_port_J1 J1

skiq_rf_port_J2 J2

skiq_rf_port_J3 J3

skiq_rf_port_J4 J4

skiq_rf_port_J5 J5

skiq_rf_port_J6 J6

skiq_rf_port_J7 J7

skiq_rf_port_J300 J300

skiq_rf_port_Jxxx_RX1 labeled Rx1
skiq_rf_port_Jxxx_TX1RX2 labeled Tx1/Rx2
skiq_rf_port_J8 J8
skiq_rf_port_max

Definition at line 793 of file sidekiq_types.h.

5.5.3 Function Documentation

5.5.3.1 EPIQ_API const char* skiq_rf_port_string (skiq_rf_port_t *rf_port*)

The skiq_rf_port_string() function returns a string representation of the passed in skiq_rf_port_t.

Since

Function added in API **v4.5.0**

Parameters

`in`	*rf_port*	RF port value

Returns

const char* string representing the RF port

5.5.3.2 EPIQ_API int32_t skiq_read_rf_port_config_avail (uint8_t *card*, bool * *p_fixed*, bool * *p_trx*)

The skiq_read_rf_port_config_avail() function determines the RF port configuration options supported by the specified Sidekiq. The RF port configuration controls the Rx/Tx capabilities for a given RF port.

See Also

skiq_read_rf_port_config
skiq_write_rf_port_config
skiq_read_rf_port_operation
skiq_write_rf_port_operation

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_fixed*	pointer indicating if fixed RF port config available
`out`	*p_trx*	pointer indicating if TRX RF port config avail

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EINVAL	reference to p_fixed or p_trx is NULL

5.5.3.3 EPIQ_API int32_t skiq_read_rf_port_config (uint8_t *card*, skiq_rf_port_config_t ∗ *p_config*)

The skiq_read_rf_port_config() function reads the current RF port configuration for the specified Sidekiq.

See Also

skiq_read_rf_port_config_avail
skiq_write_rf_port_config
skiq_read_rf_port_operation
skiq_write_rf_port_operation

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_config*	[skiq_rf_port_config_t] pointer to the current antenna configuration

Returns

int32_t status where 0=success, anything else is an error

5.5.3.4 EPIQ_API int32_t skiq_write_rf_port_config (uint8_t *card*, skiq_rf_port_config_t *config*)

The skiq_write_rf_port_config() function allows the RF port configuration of the Sidekiq card specified to be configured. To determine the available RF port configuration options, use skiq_read_rf_port_config_avail().

Note

Only particular hardware variants support certain RF port configurations.

See Also

skiq_read_rf_port_config
skiq_read_rf_port_config_avail
skiq_read_rf_port_operation
skiq_write_rf_port_operation

Parameters

in	*card*	card index of the Sidekiq of interest
in	*config*	[skiq_rf_port_config_t] RF port configuration to apply

Returns

int32_t status where 0=success, anything else is an error

5.5.3.5 EPIQ_API int32_t skiq_read_rf_port_operation (uint8_t *card*, bool ∗ *p_transmit*)

The skiq_read_rf_port_operation() function reads the operation mode of the RF port(s). If the transmit flag is set, then the port(s) are configured to transmit, otherwise it is configured for receive.

See Also

skiq_read_rf_port_config
skiq_read_rf_port_config_avail
skiq_write_rf_port_config
skiq_write_rf_port_operation

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_transmit*	pointer to flag indicating whether to transmit or receive

Returns

int32_t status where 0=success, anything else is an error

5.5.3.6 EPIQ_API int32_t skiq_write_rf_port_operation (uint8_t *card*, bool *transmit*)

The skiq_write_rf_port_operation() function sets the operation mode of the RF port(s) to either transmit or receive. If the transmit flag is set, then the port(s) are configured to transmit, otherwise it is configured for receive.

See Also

skiq_read_rf_port_config
skiq_read_rf_port_config_avail
skiq_write_rf_port_config
skiq_read_rf_port_operation

Parameters

in	*card*	card index of the Sidekiq of interest
in	*transmit*	flag indicating whether to transmit or receive

Returns

int32_t status where 0=success, anything else is an error

5.5.3.7 EPIQ_API int32_t skiq_read_rx_rf_ports_avail_for_hdl (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t ∗ *p_num_fixed_rf_ports*, skiq_rf_port_t ∗ *p_fixed_rf_port_list*, uint8_t ∗ *p_num_trx_rf_ports*, skiq_rf_port_t ∗ *p_trx_rf_port_list*)

The skiq_read_rx_rf_ports_avail_for_hdl() function reads a list of RF ports supported for the specified RX handle.

Since

Function added in API **v4.5.0**

Note

The fixed port list is only available for use when the RF port configuration is set to skiq_rf_port_config_fixed. The TRx port list is only available for use when the RF port configuration is set to skiq_rf_port_config_trx.
p_num_fixed_rf_port_list and p_trx_rf_port_list must contain at least skiq_rf_port_max number of elements.

See Also

skiq_read_rx_rf_port_for_hdl
skiq_write_rx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	RX handle of interest
out	*p_num_fixed_rf_ports*	pointer to the number of fixed RF ports available
out	*p_fixed_rf_port_list*	[skiq_rf_port_t] pointer list of fixed RF ports
out	*p_num_trx_rf_ports*	pointer to the number of TRX RF ports available
out	*p_trx_rf_port_list*	[skiq_rf_port_t] pointer list of TRX RF ports

Returns

int32_t status where 0=success, anything else is an error

5.5.3.8 EPIQ_API int32_t skiq_read_rx_rf_port_for_hdl (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rf_port_t * *p_rf_port*)

The skiq_read_rx_rf_port_for_hdl() function reads the current RF port configured for the RX handle specified.

Since

Function added in API **v4.5.0**

See Also

skiq_read_rx_rf_ports_avail_for_hdl
skiq_write_rx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	RX handle of interest

out	*p_rf_port*	[skiq_rf_port_t] pointer to the current RF port

Returns

int32_t status where 0=success, anything else is an error

5.5.3.9 EPIQ_API int32_t skiq_write_rx_rf_port_for_hdl (uint8_t *card,* skiq_rx_hdl_t *hdl,* skiq_rf_port_t *rf_port*)

The skiq_write_rx_rf_port_for_hdl() function configures the RF port for use with the RX handle.

Since

Function added in API **v4.5.0**

See Also

skiq_read_rx_rf_ports_avail_for_hdl
skiq_read_rx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	RX handle of interest
out	*rf_port*	[skiq_rf_port_t] RF port to use for hdl

Returns

int32_t status where 0=success, anything else is an error

5.5.3.10 EPIQ_API int32_t skiq_read_tx_rf_ports_avail_for_hdl (uint8_t *card,* skiq_tx_hdl_t *hdl,* uint8_t ∗ *p_num_fixed_rf_ports,* skiq_rf_port_t ∗ *p_fixed_rf_port_list,* uint8_t ∗ *p_num_trx_rf_ports,* skiq_rf_port_t ∗ *p_trx_rf_port_list*)

The skiq_read_tx_rf_ports_avail_for_hdl() function reads a list of RF ports supported for the specified TX handle.

Since

Function added in API **v4.5.0**

Note

The fixed port list is only available for use when the RF port configuration is set to skiq_rf_port_config_fixed. The TRx port list is only available for use when the RF port configuration is set to skiq_rf_port_config_trx.
p_num_fixed_rf_port_list and p_trx_rf_port_list must contain at least skiq_rf_port_max number of elements.

See Also

skiq_read_tx_rf_port_for_hdl
skiq_write_tx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	TX handle of interest
out	*p_num_fixed_rf-_ports*	pointer to the number of ports available
out	*p_fixed_rf_port-_list*	[skiq_rf_port_t] pointer list of fixed RF ports
out	*p_num_trx_rf_-ports*	pointer to the number of TRX RF ports available
out	*p_trx_rf_port_-list*	[skiq_rf_port_t] pointer list of TRX RF ports

Returns

int32_t status where 0=success, anything else is an error

5.5.3.11 EPIQ_API int32_t skiq_read_tx_rf_port_for_hdl (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_rf_port_t ∗ *p_rf_port*)

The skiq_read_tx_rf_port_for_hdl() function reads the current RF port configured for the TX handle specified.

Since

Function added in API **v4.5.0**

See Also

skiq_read_tx_rf_ports_avail_for_hdl
skiq_write_tx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	TX handle of interest
out	*p_rf_port*	[skiq_rf_port_t] pointer to the current RF port

Returns

int32_t status where 0=success, anything else is an error

5.5.3.12 EPIQ_API int32_t skiq_write_tx_rf_port_for_hdl (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_rf_port_t *rf_port*)

The skiq_write_tx_rf_port_for_hdl() function configures the RF port for use with the TX handle.

Since

Function added in API **v4.5.0**

See Also

skiq_read_tx_rf_ports_avail_for_hdl
skiq_read_tx_rf_port_for_hdl

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	TX handle of interest
out	*rf_port*	[skiq_rf_port_t] RF port to use for hdl

Returns

int32_t status where 0=success, anything else is an error

5.5.4 Variable Documentation

5.5.4.1 EPIQ_API const char∗ SKIQ_RF_PORT_STRINGS[skiq_rf_port_max]

String representation of skiq_rf_port_t enumeration.

See Also

skiq_rf_port_t

Definition at line 995 of file sidekiq_types.h.

5.6 Receive Functions and Definitions

These functions and definitions are related to configuring and exercising the receive capabilities of the Sidekiq SDR.

Classes

- struct skiq_rx_block_t

  Sidekiq Receive Block type definition for use with skiq_receive.

Macros

- #define SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS 1024

  SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS is the largest block size that can be transferred between the FPGA and the CPU in a single transaction when receiving.
- #define SKIQ_MAX_RX_BLOCK_SIZE_IN_BYTES

  The same parameter as SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS except calculated in bytes.
- #define SKIQ_RX_HEADER_SIZE_IN_WORDS 6

  The current Rx header size is 6 words but may change in the future. The metadata placed at the beginning of each IQ block. Refer to skiq_receive() for details on the formatting of the metadata.
- #define SKIQ_RX_HEADER_SIZE_IN_BYTES

  The current Rx header size, only in bytes.
- #define SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK(block_size_in_words)

  When running in packed mode, every 4 samples are 3 words of data. SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK converts from number of words to number of samples when running in packed mode.
- #define SKIQ_NUM_WORDS_IN_PACKED_BLOCK(num_packed_samples)

  When running in packed mode, every 3 words of data contain 4 samples. SKIQ_NUM_WORDS_IN_PACKED_BLO-CK converts from the number of samples to the number of words needed to hold the number of unpacked samples. The SKIQ_NUM_WORDS_IN_PACKED_BLOCK macro rounds up by adding one less than the denominator (the number of bytes in a word: 4) prior to performing the integer division.
- #define SKIQ_RX_NUM_PACKETS_IN_RING_BUFFER (2048)

  The number of packets in the ring buffer is the number of packets that can be buffered and not yet received prior to the packets getting overwritten.
- #define RX_TRANSFER_NO_WAIT (0)

  Option for timeout_us argument of skiq_set_rx_transfer_timeout() to return immediately, regardless as to whether or not samples are available. Effectively results in a non-blocking skiq_receive() call and the return code is set accordingly.
- #define RX_TRANSFER_WAIT_FOREVER (-1)

  Option for timeout_us argument of skiq_set_rx_transfer_timeout() to block forever until samples are available. Effectively results in a blocking skiq_receive() call with no timeout. Use with caution (or don't use at all)
- #define RX_TRANSFER_WAIT_NOT_SUPPORTED (-2)

  *Possible value for **p_timeout_us** argument of skiq_get_rx_transfer_timeout() to indicate that blocking skiq_-receive() is not supported by the card and/or its currently configured transport layer (skiq_xport_type_t).*
- #define SKIQ_RX_BLOCK_INITIALIZER_BY_BYTES(var_name, data_size_in_bytes)
- #define SKIQ_RX_BLOCK_INITIALIZER_BY_WORDS(var_name, data_size_in_words)
- #define SKIQ_RX_BLOCK_INITIALIZER(var_name)

Enumerations

- enum skiq_data_src_t { skiq_data_src_iq =0, skiq_data_src_counter }

  An Rx interface is typically configured to generate complex I/Q samples. However, there are test cases where it is useful to have the I/Q data replaced with an incrementing counter. This is treated as a different "data source", which can be configured by the user at run-time before an Rx interface is started.
- enum skiq_rx_stream_mode_t { skiq_rx_stream_mode_high_tput =0, skiq_rx_stream_mode_low_latency, skiq_rx_stream_mode_balanced, skiq_rx_stream_mode_end }

  Sidekiq supports three different receive stream modes that change the relative IQ sample block latency (skiq_rx_block_t) between the FPGA and host CPU. The skiq_rx_stream_mode_high_tput setting is business as usual and provides the same receive latency that exists in the previous releases of libsidekiq. The skiq_rx_stream_mode_low_latency setting provides a smaller block of IQ samples from skiq_receive() more often and effectively lowers the latency from RF reception to host CPU. The skiq_rx_stream_mode_balanced is a compromise between the high_tput and low_latency modes which has a reduced overall throughput relative to high_tput but results in a larger number of samples per packet than the low_latency mode.
- enum skiq_trigger_src_t { skiq_trigger_src_immediate =0, skiq_trigger_src_1pps, skiq_trigger_src_synced }

  A trigger source may be specified when starting or stopping multiple handle streaming. The trigger may be specified as 'immediate' and happens without any synchronization between handles. It may also be specified as '1PPS' so all specified handles would start streaming synchronized on a PPS edge. If the FPGA bitstream supports it (>= 3.11.0), a value of skiq_trigger_src_synced causes all specified handles to start or stop streaming immediately, but also synchronized (RF timestamps are aligned). A similar application may be used when stopping multiple handles from streaming.
- enum skiq_rx_hdl_t {
  skiq_rx_hdl_A1 =0, skiq_rx_hdl_A2 =1, skiq_rx_hdl_B1 =2, skiq_rx_hdl_B2 =3,
  skiq_rx_hdl_C1 =4, skiq_rx_hdl_D1 =5, skiq_rx_hdl_end }

  Sidekiq supports several Rx interface handles. The skiq_rx_hdl_t enum is used to define the different Rx interface handles.
- enum skiq_rx_gain_t { skiq_rx_gain_manual = 0, skiq_rx_gain_auto }

  Rx gain can be controlled either manually or automatically. The skiq_rx_gain_t enum is used to specify the mode of gain control.
- enum skiq_rx_attenuation_mode_t { skiq_rx_attenuation_mode_manual =0, skiq_rx_attenuation_mode_noise_figure, skiq_rx_attenuation_mode_normalized }

  Rx attenuation mode.
- enum skiq_chan_mode_t { skiq_chan_mode_single =0, skiq_chan_mode_dual }

  Sidekiq can run either in single Rx or dual channel Rx mode. The skiq_chan_mode_t enum is used to specify the Rx/Tx channel mode.
- enum skiq_rx_fir_gain_t { skiq_rx_fir_gain_neg_12 =3, skiq_rx_fir_gain_neg_6 =2, skiq_rx_fir_gain_0 =1, skiq_rx_fir_gain_6 =0 }

  Rx FIR filter gain settings, applied to the Rx FIR used in the Rx channel bandwidth configuration.
- enum skiq_rx_status_t {
  skiq_rx_status_success = 0, skiq_rx_status_no_data = -1, skiq_rx_status_error_generic = -6, skiq_rx_status_error_overrun = -11,
  skiq_rx_status_error_packet_malformed = -12, skiq_rx_status_error_card_not_active = -19, skiq_rx_status_error_not_streaming = -29 }

  Possible return codes from skiq_receive.
- enum skiq_rx_cal_mode_t { skiq_rx_cal_mode_auto =0, skiq_rx_cal_mode_manual }

  RX Calibration Mode.
- enum skiq_rx_cal_type_t { skiq_rx_cal_type_none = 0x00000000, skiq_rx_cal_type_dc_offset = 0x00000001, skiq_rx_cal_type_quadrature = 0x00000002 }

  RX Calibration Types.

Functions

- EPIQ_API int32_t skiq_read_rx_streaming_handles (uint8_t card, skiq_rx_hdl_t ∗p_hdls_streaming, uint8_t ∗p_num_hdls)
- EPIQ_API int32_t skiq_read_rx_stream_handle_conflict (uint8_t card, skiq_rx_hdl_t hdl_to_stream, skiq_rx_hdl_t ∗p_conflicting_hdls, uint8_t ∗p_num_hdls)
- EPIQ_API int32_t skiq_start_rx_streaming (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_immediate (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_synced (uint8_t card, skiq_rx_hdl_t handles[ ], uint8-_t nr_handles)
- EPIQ_API int32_t skiq_start_rx_streaming_on_1pps (uint8_t card, skiq_rx_hdl_t hdl, uint64_t sys_-timestamp)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_on_trigger (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, skiq_trigger_src_t trigger, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_stop_rx_streaming (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_immediate (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_synced (uint8_t card, skiq_rx_hdl_t handles[ ], uint8-_t nr_handles)
- EPIQ_API int32_t skiq_stop_rx_streaming_on_1pps (uint8_t card, skiq_rx_hdl_t hdl, uint64_t sys_-timestamp)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_on_trigger (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, skiq_trigger_src_t trigger, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_read_chan_mode (uint8_t card, skiq_chan_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_chan_mode (uint8_t card, skiq_chan_mode_t mode)
- EPIQ_API int32_t skiq_write_rx_preselect_filter_path (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t path)
- EPIQ_API int32_t skiq_read_rx_preselect_filter_path (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t ∗p_-path)
- EPIQ_API int32_t skiq_read_rx_overload_state (uint8_t card, skiq_rx_hdl_t hdl, uint8_t ∗p_overload)
- EPIQ_API int32_t skiq_read_rx_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t ∗p_freq, double ∗p-_actual_freq)
- EPIQ_API int32_t skiq_write_rx_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t freq)
- EPIQ_API int32_t skiq_read_rx_sample_rate (uint8_t card, skiq_rx_hdl_t hdl, uint32_t ∗p_rate, double ∗p_actual_rate)
- EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32-_t rate, uint32_t bandwidth)
- EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth_multi (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, uint32_t rate[ ], uint32_t bandwidth[ ])
- EPIQ_API int32_t skiq_read_rx_sample_rate_and_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t ∗p_rate, double ∗p_actual_rate, uint32_t ∗p_bandwidth, uint32_t ∗p_actual_bandwidth)
- EPIQ_API int32_t skiq_set_rx_transfer_timeout (const uint8_t card, const int32_t timeout_us)
- EPIQ_API int32_t skiq_get_rx_transfer_timeout (const uint8_t card, int32_t ∗p_timeout_us)
- EPIQ_API skiq_rx_status_t skiq_receive (uint8_t card, skiq_rx_hdl_t ∗p_hdl, skiq_rx_block_t ∗∗pp_-block, uint32_t ∗p_data_len)
- EPIQ_API int32_t skiq_read_rx_gain_index_range (uint8_t card, skiq_rx_hdl_t hdl, uint8_t ∗p_gain_-index_min, uint8_t ∗p_gain_index_max)
- EPIQ_API int32_t skiq_write_rx_gain (uint8_t card, skiq_rx_hdl_t hdl, uint8_t gain_index)
- EPIQ_API int32_t skiq_read_rx_gain (uint8_t card, skiq_rx_hdl_t hdl, uint8_t ∗p_gain_index)
- EPIQ_API int32_t skiq_read_rx_gain_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_gain_t ∗p_gain_-mode)

- EPIQ_API int32_t skiq_write_rx_gain_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_gain_t gain_mode)
- EPIQ_API int32_t skiq_write_rx_attenuation_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_attenuation_mode_t mode)
- EPIQ_API int32_t skiq_read_rx_attenuation_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_attenuation_mode_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_attenuation (uint8_t card, skiq_rx_hdl_t hdl, uint16_t attenuation)
- EPIQ_API int32_t skiq_read_rx_attenuation (uint8_t card, skiq_rx_hdl_t hdl, uint16_t *p_attenuation)
- EPIQ_API int32_t skiq_write_rx_stream_mode (uint8_t card, skiq_rx_stream_mode_t stream_mode)
- EPIQ_API int32_t skiq_read_rx_stream_mode (uint8_t card, skiq_rx_stream_mode_t *p_stream_mode)
- EPIQ_API int32_t skiq_write_rx_dc_offset_corr (uint8_t card, skiq_rx_hdl_t hdl, bool enable)
- EPIQ_API int32_t skiq_read_rx_dc_offset_corr (uint8_t card, skiq_rx_hdl_t hdl, bool *p_enable)
- EPIQ_API int32_t skiq_write_rx_fir_gain (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_fir_gain_t gain)
- EPIQ_API int32_t skiq_read_rx_fir_gain (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_fir_gain_t *p_gain)
- EPIQ_API int32_t skiq_read_num_rx_chans (uint8_t card, uint8_t *p_num_rx_chans)
- EPIQ_API int32_t skiq_read_rx_iq_resolution (uint8_t card, uint8_t *p_adc_res)
- EPIQ_API int32_t skiq_read_rx_LO_freq_range (uint8_t card, uint64_t *p_max, uint64_t *p_min)
- EPIQ_API int32_t skiq_read_max_rx_LO_freq (uint8_t card, uint64_t *p_max)
- EPIQ_API int32_t skiq_read_min_rx_LO_freq (uint8_t card, uint64_t *p_min)
- EPIQ_API int32_t skiq_read_rx_block_size (uint8_t card, skiq_rx_stream_mode_t stream_mode)
- EPIQ_API int32_t skiq_read_rx_cal_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_cal_mode_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_cal_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_cal_mode_t mode)
- EPIQ_API int32_t skiq_run_rx_cal (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_rx_cal_type_mask (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_cal_mask)
- EPIQ_API int32_t skiq_write_rx_cal_type_mask (uint8_t card, skiq_rx_hdl_t hdl, uint32_t cal_mask)
- EPIQ_API int32_t skiq_read_rx_cal_types_avail (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_cal_mask)
- EPIQ_API int32_t skiq_read_rx_analog_filter_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_bandwidth)
- EPIQ_API int32_t skiq_write_rx_analog_filter_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t bandwidth)

Variables

- EPIQ_API const char * SKIQ_RX_STREAM_MODE_STRINGS [skiq_rx_stream_mode_end]

  String representation of skiq_rx_stream_mode_t.

5.6.1 Detailed Description

These functions and definitions are related to configuring and exercising the receive capabilities of the Sidekiq SDR.

5.6.2 Macro Definition Documentation

5.6.2.1 #define SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS 1024

SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS is the largest block size that can be transferred between the FPGA and the CPU in a single transaction when receiving.

Definition at line 347 of file sidekiq_api.h.

5.6.2.2 #define SKIQ_MAX_RX_BLOCK_SIZE_IN_BYTES

The same parameter as SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS except calculated in bytes.

Definition at line 351 of file sidekiq_api.h.

5.6.2.3 #define SKIQ_RX_HEADER_SIZE_IN_WORDS 6

The current Rx header size is 6 words but may change in the future. The metadata placed at the beginning of each IQ block. Refer to skiq_receive() for details on the formatting of the metadata.

Definition at line 356 of file sidekiq_api.h.

5.6.2.4 #define SKIQ_RX_HEADER_SIZE_IN_BYTES

The current Rx header size, only in bytes.

Definition at line 359 of file sidekiq_api.h.

5.6.2.5 #define SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK(*block_size_in_words*)

When running in packed mode, every 4 samples are 3 words of data. SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK converts from number of words to number of samples when running in packed mode.

Definition at line 364 of file sidekiq_api.h.

5.6.2.6 #define SKIQ_NUM_WORDS_IN_PACKED_BLOCK(*num_packed_samples*)

When running in packed mode, every 3 words of data contain 4 samples. SKIQ_NUM_WORDS_IN_PACKED_BLOCK converts from the number of samples to the number of words needed to hold the number of unpacked samples. The SKIQ_NUM_WORDS_IN_PACKED_BLOCK macro rounds up by adding one less than the denominator (the number of bytes in a word: 4) prior to performing the integer division.

For example, if a user wants 5 packed samples, then 4 words of data must be considered when unpacking. Packed samples occupy 24 bits and words are 32 bits

5 x 24 bits < 4 x 32 bits == 120 bits < 128 bits

SKIQ_NUM_WORDS_IN_PACKED_BLOCK(5) = ((5 ∗ 3) + 3) / 4 = (15 + 3) / 4 = 18 / 4 = 4

Another example is if a user wants 1906250 packed samples, then 1429688 words of data must be considered when unpacking.

1906250 x 24 bits < 1429688 x 32 bits == 45750000 bits < 45750016 bits

SKIQ_NUM_WORDS_IN_PACKED_BLOCK(1906250) = ((1906250 ∗ 3) + 3) / 4 = (5718750 + 3) / 4 = 5718753 / 4 = 1429688

Definition at line 393 of file sidekiq_api.h.

5.6.2.7 #define SKIQ_RX_NUM_PACKETS_IN_RING_BUFFER (2048)

The number of packets in the ring buffer is the number of packets that can be buffered and not yet received prior to the packets getting overwritten.

Deprecated As of libsidekiq v4.13, this value is no longer guaranteed to be accurate as the value can change based upon the configuration of the PCI DMA Driver kernel module.

Definition at line 405 of file sidekiq_api.h.

5.6.2.8 #define RX_TRANSFER_NO_WAIT (0)

Option for timeout_us argument of skiq_set_rx_transfer_timeout() to return immediately, regardless as to whether or not samples are available. Effectively results in a non-blocking skiq_receive() call and the return code is set accordingly.

Definition at line 629 of file sidekiq_api.h.

5.6.2.9 #define RX_TRANSFER_WAIT_FOREVER (-1)

Option for timeout_us argument of skiq_set_rx_transfer_timeout() to block forever until samples are available. Effectively results in a blocking skiq_receive() call with no timeout. Use with caution (or don't use at all)

- a failure to transfer samples will result in the calling thread being blocked indefinitely.

Definition at line 637 of file sidekiq_api.h.

5.6.2.10 #define RX_TRANSFER_WAIT_NOT_SUPPORTED (-2)

Possible value for **p_timeout_us** argument of skiq_get_rx_transfer_timeout() to indicate that blocking skiq_receive() is not supported by the card and/or its currently configured transport layer (skiq_xport_type_t).

Definition at line 644 of file sidekiq_api.h.

5.6.2.11 #define SKIQ_RX_BLOCK_INITIALIZER_BY_BYTES(*var_name*, *data_size_in_bytes*)

Sidekiq Receive Block static initializer, user specifies the number of desired bytes.

Note

Sidekiq Receive Blocks statically allocated should be typecast to a skiq_rx_block_t reference when calling skiq_receive to avoid compiler warnings

Since

MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name
in	*data_size_in_bytes*	desired payload size (bytes)

Definition at line 1187 of file sidekiq_types.h.

5.6.2.12 #define SKIQ_RX_BLOCK_INITIALIZER_BY_WORDS(*var_name,* *data_size_in_words*)

Sidekiq Receive Block static initializer, user specifies the number of desired words

Note

Sidekiq Receive Blocks statically allocated should be typecast to a skiq_rx_block_t reference when calling skiq_receive to avoid compiler warnings

Since

MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name
in	*data_size_in_words*	desired payload size (words)

Definition at line 1207 of file sidekiq_types.h.

5.6.2.13 #define SKIQ_RX_BLOCK_INITIALIZER(*var_name*)

Sidekiq Receive Block static initializer, allocates the maximum receive block size

Note

Sidekiq Receive Blocks statically allocated should be typecast to a skiq_rx_block_t reference when calling skiq_receive to avoid compiler warnings

Since

MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name

Definition at line 1226 of file sidekiq_types.h.

5.6.3 Enumeration Type Documentation

5.6.3.1 enum skiq_data_src_t

An Rx interface is typically configured to generate complex I/Q samples. However, there are test cases where it is useful to have the I/Q data replaced with an incrementing counter. This is treated as a different "data source", which can be configured by the user at run-time before an Rx interface is started.

See Also

skiq_read_rx_data_src
skiq_write_rx_data_src

Enumerator

skiq_data_src_iq

skiq_data_src_counter

Definition at line 263 of file sidekiq_types.h.

5.6.3.2 enum skiq_rx_stream_mode_t

Sidekiq supports three different receive stream modes that change the relative IQ sample block latency (skiq_rx_block_t) between the FPGA and host CPU. The skiq_rx_stream_mode_high_tput setting is business as usual and provides the same receive latency that exists in the previous releases of libsidekiq. The skiq_rx_stream_mode_low_latency setting provides a smaller block of IQ samples from skiq_receive() more often and effectively lowers the latency from RF reception to host CPU. The skiq_rx_stream_mode_balanced is a compromise between the high_tput and low_latency modes which has a reduced overall throughput relative to high_tput but results in a larger number of samples per packet than the low_latency mode.

Attention

Since skiq_rx_stream_mode_low_latency setting delivers smaller blocks of IQ samples (with metadata) more often, it is only effective up to 8-10Msps (∼3Msps on 32-bit ARM systems). The user will encounter timestamp gaps if using this mode in conjunction with sample rates above this limitation.

Since

definition added in **v4.6.0**, skiq_rx_stream_mode_balanced added in **v4.7.0** skiq_rx_stream_mode_low_latency requires FPGA **v3.9.0** or later

See Also

skiq_read_rx_stream_mode
skiq_write_rx_stream_mode

Enumerator

skiq_rx_stream_mode_high_tput

skiq_rx_stream_mode_low_latency

skiq_rx_stream_mode_balanced

skiq_rx_stream_mode_end

Definition at line 333 of file sidekiq_types.h.

5.6.3.3 enum skiq_trigger_src_t

A trigger source may be specified when starting or stopping multiple handle streaming. The trigger may be specified as ’immediate’ and happens without any synchronization between handles. It may also be specified as ’1PPS’ so all specified handles would start streaming synchronized on a PPS edge. If the FPGA bitstream supports it (>= 3.11.0), a value of skiq_trigger_src_synced causes all specified handles to start or stop streaming immediately, but also synchronized (RF timestamps are aligned). A similar application may be used when stopping multiple handles from streaming.

Note

Presently limited to receive handles

Since

Definition added in **v4.5.0**, skiq_trigger_src_synced added in **v4.8.0**

See Also

skiq_start_rx_streaming_multi_on_trigger
skiq_stop_rx_streaming_multi_on_trigger

Enumerator

skiq_trigger_src_immediate
skiq_trigger_src_1pps
skiq_trigger_src_synced

Definition at line 360 of file sidekiq_types.h.

5.6.3.4 enum skiq_rx_hdl_t

Sidekiq supports several Rx interface handles. The skiq_rx_hdl_t enum is used to define the different Rx interface handles.

Enumerator

skiq_rx_hdl_A1
skiq_rx_hdl_A2
skiq_rx_hdl_B1
skiq_rx_hdl_B2
skiq_rx_hdl_C1
skiq_rx_hdl_D1
skiq_rx_hdl_end

Definition at line 373 of file sidekiq_types.h.

5.6.3.5 enum skiq_rx_gain_t

Rx gain can be controlled either manually or automatically. The skiq_rx_gain_t enum is used to specify the mode of gain control.

See Also

skiq_read_rx_gain_mode
skiq_write_rx_gain_mode

Enumerator

skiq_rx_gain_manual
skiq_rx_gain_auto

Definition at line 527 of file sidekiq_types.h.

5.6.3.6 enum skiq_rx_attenuation_mode_t

Rx attenuation mode.

Attention

This is only supported for `Sidekiq X2`.

See Also

skiq_read_rx_attenuation
skiq_read_rx_attenuation_mode
skiq_write_rx_attenuation
skiq_write_rx_attenuation_mode

Enumerator

skiq_rx_attenuation_mode_manual User is responsible for writing Rx attenuation value.

skiq_rx_attenuation_mode_noise_figure Software automatically configures attenuation to optimize for the best noise figure across all frequencies.

skiq_rx_attenuation_mode_normalized Software automatically configures attenuation to optimize for equal gain response across all frequencies.

Definition at line 544 of file sidekiq_types.h.

5.6.3.7 enum skiq_chan_mode_t

Sidekiq can run either in single Rx or dual channel Rx mode. The skiq_chan_mode_t enum is used to specify the Rx/Tx channel mode.

Warning

Dual channel mode is only supported with SKIQ-001 hardware.

See Also

skiq_read_chan_mode
skiq_write_chan_mode

Enumerator

skiq_chan_mode_single only A1 is enabled for Rx/Tx

skiq_chan_mode_dual both A1 and A2 are enabled for Rx/Tx

Definition at line 574 of file sidekiq_types.h.

5.6.3.8 enum skiq_rx_fir_gain_t

Rx FIR filter gain settings, applied to the Rx FIR used in the Rx channel bandwidth configuration.

See Also

skiq_read_rx_fir_gain
skiq_write_rx_fir_gain

Enumerator

skiq_rx_fir_gain_neg_12 designates a receive FIR gain of -12 dB

skiq_rx_fir_gain_neg_6 designates a receive FIR gain of -6 dB

skiq_rx_fir_gain_0 designates a receive FIR gain of 0 dB

skiq_rx_fir_gain_6 designates a receive FIR gain of +6 dB

Definition at line 674 of file sidekiq_types.h.

5.6.3.9 enum skiq_rx_status_t

Possible return codes from skiq_receive.

Enumerator

skiq_rx_status_success new data is available

skiq_rx_status_no_data no new data is ready

skiq_rx_status_error_generic a generic error was encountered when trying to receive

skiq_rx_status_error_overrun an overrun occurred. An overrun occurs when the FPGA streams data faster than software retrieves it, resulting in the data not yet retrieved by software to be overwritten. This condition is reset upon each skiq_receive call.

skiq_rx_status_error_packet_malformed packet was incorrectly structured/formatted

skiq_rx_status_error_card_not_active requested card not active in current session

skiq_rx_status_error_not_streaming no receive handles streaming, cannot receive a block

Definition at line 735 of file sidekiq_types.h.

5.6.3.10 enum skiq_rx_cal_mode_t

RX Calibration Mode.

See Also

skiq_read_rx_cal_mode
skiq_write_rx_cal_mode
skiq_run_rx_cal

Enumerator

skiq_rx_cal_mode_auto automatically run RX calibration algorithms

skiq_rx_cal_mode_manual do not automatically run the RX algorithms

Definition at line 832 of file sidekiq_types.h.

5.6.3.11 enum skiq_rx_cal_type_t

RX Calibration Types.

See Also

skiq_write_rx_cal_type_mask
skiq_read_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Enumerator

skiq_rx_cal_type_none
skiq_rx_cal_type_dc_offset
skiq_rx_cal_type_quadrature

Definition at line 845 of file sidekiq_types.h.

5.6.4 Function Documentation

5.6.4.1 EPIQ_API int32_t skiq_read_rx_streaming_handles (uint8_t *card*, skiq_rx_hdl_t ∗ *p_hdls_streaming*, uint8_t ∗ *p_num_hdls*)

The skiq_read_rx_streaming_handles() function is responsible for providing a list of RX handles currently streaming.

Since

Function added in **v4.9.0**

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_hdls_-streaming*	[skiq_rx_hdl_t] array of handles currently streaming
out	*p_num_hdls*	pointer of where to store number of handles in streaming list

Returns

int32_t

Return values

0	p_hdls_streaming populated with RX handles currently streaming
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
non-zero	Unspecified error occurred

5.6.4.2 EPIQ_API int32_t skiq_read_rx_stream_handle_conflict (uint8_t *card*, skiq_rx_hdl_t *hdl_to_stream*, skiq_rx_hdl_t ∗ *p_conflicting_hdls*, uint8_t ∗ *p_num_hdls*)

The skiq_read_rx_stream_handle_conflict() function is responsible for providing a list of RX handles that cannot be streaming simultaneous to the handle specified. If streaming is requested with a conflicting handle, the stream cannot be started.

Since

Function added in **v4.9.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl_to_stream*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_conflicting_-hdls*	[skiq_rx_hdl_t] array of handles that conflict. Must be large enough to contain skiq_rx_hdl_end elements.
out	*p_num_hdls*	pointer of where to store number of handles in conflict list

Returns

int32_t

Return values

0	p_hdls_streaming populated with RX handles currently streaming
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EINVAL	Error occurred reading conflicting handles
non-zero	other error occurred

5.6.4.3 EPIQ_API int32_t skiq_start_rx_streaming (uint8_t *card*, skiq_rx_hdl_t *hdl*)

The skiq_start_rx_streaming() function is responsible for starting the flow of data between the FPGA and the CPU. This function triggers the FPGA to start receiving data and transferring it to the CPU. A continuous flow of packets will be transferred from the FPGA to the CPU until the user app calls skiq_stop_rx_streaming(). These packets will be received by the user app by calling skiq_receive(), which returns one packet at a time.

This function call is functionally equivalent to:

```
skiq_start_rx_streaming_multi_on_trigger( card, &hdl, 1,
                                          skiq_trigger_src_immediate,
                                          0 )
```

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface

Returns

int32_t

Return values

0	successful start streaming for handle specified
-ERANGE	Requested card index is out of range

-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-EBUSY	One of the specified handles is already streaming
-EBUSY	A conflicting handle is already streaming
-ENOTSUP	Configured RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	Configured RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode
-EIO	Failed to start streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOSYS	Transport does not support FPGA register access
non-zero	An unspecified error occurred

5.6.4.4 EPIQ_API int32_t skiq_start_rx_streaming_multi_immediate (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*)

The skiq_start_rx_streaming_multi_immediate() function allows a user to start multiple receive streams immediately (not necessarily timestamp-synchronized depending on FPGA support and library support).

Warning

If one of the receive handles is already streaming then this function returns an error.

This function call is functionally equivalent to:

```
skiq_start_rx_streaming_multi_on_trigger( card, handles, nr_handles,
                                          skiq_trigger_src_immediate,
                                          0 )
```

Since

Function added in **v4.5.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to start streaming
in	*nr_handles*	the number of entries in handles[]

Returns

int32_t

Return values

0	successful start streaming for handle specified

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles $<$ 1, etc)
-EBUSY	One of the specified handles is already streaming
-EBUSY	A conflicting handle is already streaming
-ENOTSUP	Configured RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	Configured RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode
-EIO	Failed to start streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOSYS	Transport does not support FPGA register access
non-zero	An unspecified error occurred

5.6.4.5 EPIQ_API int32_t skiq_start_rx_streaming_multi_synced (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*)

The skiq_start_rx_streaming_multi_synced() function allows a user to start multiple receive streams immediately and with timestamp synchronization (not necessarily phase coherent however).

Warning

If one of the receive handles is already streaming then this function returns an error.

Attention

Not all Sidekiq products support the use of this function.

Since

Function added in **v4.9.0**, requires FPGA bitstream **v3.11.0** or greater

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to start streaming
in	*nr_handles*	the number of entries in handles[]

Returns

int32_t

Return values

0	successful start streaming for handle specified

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-EBUSY	One of the specified handles is already streaming
-EBUSY	A conflicting handle is already streaming
-ENOTSUP	Configured RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	Configured RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode
-EIO	Failed to start streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOSYS	Transport does not support FPGA register access
-ENOTSUP	the skiq_trigger_src_synced trigger source is not supported for the given Sidekiq product or FPGA bitstream
non-zero	An unspecified error occurred

5.6.4.6 EPIQ_API int32_t skiq_start_rx_streaming_on_1pps (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *sys_timestamp*)

The skiq_start_rx_streaming_on_1pps() function is identical to the skiq_start_rx_streaming() with exception of when the data stream starts to flow. When calling this function, the data does not begin to flow until the rising 1PPS edge after the system timestamp specified has occurred. If a timestamp of 0 is provided, then the next 1PPS edge will begin the data flow. This function blocks until the data starts flowing.

This function call is functionally equivalent to:

```
skiq_start_rx_streaming_multi_on_trigger( card, &hdl, 1,
                                          skiq_trigger_src_1pps,
                                          sys_timestamp )
```

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*sys_timestamp*	system timestamp after the next 1PPS will begin the data flow

Returns

int32_t

Return values

0	successful start streaming for handle specified
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified

-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-EBUSY	One of the specified handles is already streaming
-EBUSY	A conflicting handle is already streaming
-ENOTSUP	Configured RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	Configured RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode
-EIO	Failed to start streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOSYS	Transport does not support FPGA register access
non-zero	An unspecified error occurred

5.6.4.7 EPIQ_API int32_t skiq_start_rx_streaming_multi_on_trigger (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*, skiq_trigger_src_t *trigger*, uint64_t *sys_timestamp*)

The skiq_start_rx_streaming_multi_on_trigger() function allows a user to start multiple receive streams after the specified trigger occurs.

Warning

If one of the receive handles is already streaming then this function returns an error.

Attention

If skiq_trigger_src_1pps is used as a trigger then this function will **block** until the 1PPS edge occurs.

Since

Function added in **v4.5.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to start streaming
in	*nr_handles*	the number of entries in handles[]
in	*trigger*	[skiq_trigger_src_t] type of trigger to use
in	*sys_timestamp*	System Timestamp after the next positive trigger will begin the data flow

Returns

int32_t

Return values

0	successful start streaming for handle specified
-ERANGE	Requested card index is out of range

-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-EBUSY	One of the specified handles is already streaming
-EBUSY	A conflicting handle is already streaming
-ENOTSUP	Configured RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	Configured RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode
-EIO	Failed to start streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOSYS	Transport does not support FPGA register access
non-zero	An unspecified error occurred

5.6.4.8 EPIQ_API int32_t skiq_stop_rx_streaming (uint8_t *card*, skiq_rx_hdl_t *hdl*)

The skiq_stop_rx_streaming() function is responsible for stopping the streaming of data between the FPGA and the CPU. This function can only be called after an interface has previously started streaming.

This function call is functionally equivalent to:

```
skiq_stop_rx_streaming_multi_on_trigger( card, &hdl, 1,
                                         skiq_trigger_src_immediate,
                                         0 )
```

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-ENODEV	One of the specified handles is not currently streaming
-EIO	Failed to stop streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
non-zero	Unspecified error occurred

5.6.4.9 EPIQ_API int32_t skiq_stop_rx_streaming_multi_immediate (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*)

The skiq_stop_rx_streaming_multi_immediate() function allows a user to stop multiple receive streams immediately (not necessarily timestamp-synchronized depending on FPGA support and library support).

Warning

If one of the receive handles is not streaming then this function returns an error.

This function call is functionally equivalent to:

```
skiq_stop_rx_streaming_multi_on_trigger( card, handles, nr_handles,
                                         skiq_trigger_src_immediate,
                                         0 )
```

Since

Function added in **v4.5.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to start streaming
in	*nr_handles*	the number of entries in handles[]

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles $<$ 1, etc)
-ENODEV	One of the specified handles is not currently streaming
-EIO	Failed to stop streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
non-zero	Unspecified error occurred

5.6.4.10 EPIQ_API int32_t skiq_stop_rx_streaming_multi_synced (uint8_t *card,* skiq_rx_hdl_t *handles[ ],* uint8_t *nr_handles*)

The skiq_stop_rx_streaming_multi_synced() function allows a user to stop multiple receive streams immediately and with timestamp synchronization (not necessarily phase coherent however).

Warning

If one of the receive handles is not streaming then this function returns an error.

Attention

Not all Sidekiq products support this function.

Since

Function added in **v4.9.0**, requires FPGA bitstream **v3.11.0** or greater

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to start streaming
in	*nr_handles*	the number of entries in handles[]

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles $<$ 1, etc)
-ENODEV	One of the specified handles is not currently streaming
-EIO	Failed to stop streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOTSUP	the skiq_trigger_src_synced trigger source is not supported for the given Sidekiq product or FPGA bitstream
non-zero	Unspecified error occurred

5.6.4.11 EPIQ_API int32_t skiq_stop_rx_streaming_on_1pps (uint8_t *card,* skiq_rx_hdl_t *hdl,* uint64_t *sys_timestamp*)

The skiq_stop_rx_streaming_on_1pps() function stops the data from flowing on the rising edge of the 1PPS after the timestamp specified. If a timestamp of 0 is provided, then the next 1PPS edge will stop the data flow. This function blocks until the data stream has been stopped.

Note

this stops the data at the FPGA. However, there will be data remaining in the internal FIFOs, so skiq_receive() should continue to be called until no data remains. Once that is complete, the skiq_stop_rx_streaming() function should be called to finalize the disabling of the data flow.

This function call is functionally equivalent to:

```
skiq_stop_rx_streaming_multi_on_trigger( card, &hdl, 1,
                                         skiq_trigger_src_1pps,
                                         sys_timestamp )
```

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*sys_timestamp*	system timestamp after the next 1PPS will stop the data flow

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-ENODEV	One of the specified handles is not currently streaming
-EIO	Failed to stop streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
non-zero	Unspecified error occurred

5.6.4.12 EPIQ_API int32_t skiq_stop_rx_streaming_multi_on_trigger (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*, skiq_trigger_src_t *trigger*, uint64_t *sys_timestamp*)

The skiq_stop_rx_streaming_multi_on_trigger() function allows a user to stop multiple receive streams after the specified trigger occurs.

Warning

If one of the receive handles is not streaming then this function returns an error.

Attention

If skiq_trigger_src_1pps is used as a trigger then this function will **block** until the 1PPS edge occurs.

Since

Function added in **v4.5.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[array of skiq_rx_hdl_t] the receive handles to stop streaming
in	*nr_handles*	the number of entries in handles[]
in	*trigger*	[skiq_trigger_src_t] type of trigger to use
in	*sys_timestamp*	System Timestamp after the next positive trigger will stop the data flow

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized

-EDOM	Invalid RX handle specified
-EINVAL	Invalid parameter passed (nr_handles < 1, etc)
-ENODEV	One of the specified handles is not currently streaming
-EIO	Failed to stop streaming for given transport
-ECOMM	Communication error occurred transacting with FPGA registers
-ENOTSUP	the skiq_trigger_src_synced trigger source is not supported for the given Sidekiq product or FPGA bitstream
non-zero	Unspecified error occurred

5.6.4.13 EPIQ_API int32_t skiq_read_chan_mode (uint8_t *card*, skiq_chan_mode_t ∗ *p_mode*)

The skiq_read_chan_mode() function is responsible for returning the current Rx channel mode (skiq_chan_mode_t) setting.

See Also

skiq_write_chan_mode

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_mode*	pointer to where to store the Rx channel mode setting

Returns

int32_t status where 0=success, anything else is an error

5.6.4.14 EPIQ_API int32_t skiq_write_chan_mode (uint8_t *card*, skiq_chan_mode_t *mode*)

The skiq_write_chan_mode() function is responsible for configuring the channel mode. If only A1 is needed for receiving or if transmit is being used it is recommended to configure the mode to skiq_chan_mode_single. If A2 is being used as a receiver or if both A1 and A2 are being used as receivers, than the mode should be configured to skiq_chan_mode_dual.

See Also

skiq_read_chan_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*mode*	specifies the Rx channel mode setting

Returns

int32_t status where 0=success, anything else is an error

5.6.4.15 EPIQ_API int32_t skiq_write_rx_preselect_filter_path (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_filt_t *path*)

The skiq_write_rx_preselect_filter_path() function is responsible for selecting from any skiq_filt_t value appropriate for the Sidekiq hardware on the specified Rx interface.

Note

Not all filter options are available for hardware variants. Users may use skiq_read_rx_filters_avail() to determine RF filter path available for a given Sidekiq card.

See Also

skiq_read_rx_filters_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*path*	an enum indicating which path is being requested

Returns

int32_t status where 0=success, anything else is an error

5.6.4.16 EPIQ_API int32_t skiq_read_rx_preselect_filter_path (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_filt_t ∗ *p_path*)

The skiq_read_rx_preselect_filter_path() function is responsible for returning the currently selected RF filter path (of type skiq_filt_t) on the specified Rx interface.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_path*	a pointer to where the current value of the filter path should be written

Returns

int32_t status where 0=success, anything else is an error

5.6.4.17 EPIQ_API int32_t skiq_read_rx_overload_state (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t ∗ *p_overload*)

The skiq_read_rx_overload_state() function is responsible for reporting the overload state of the specified Rx interface. An overload condition is detected when an RF input in excess of 0dBm is detected. If an overload condition is detected, the state is 1, otherwise it is 0.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_overload*	a pointer to where to store the overload state.

Returns

int32_t status where 0=success, anything else is an error

5.6.4.18 EPIQ_API int32_t skiq_read_rx_LO_freq (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t * *p_freq*, double * *p_actual_freq*)

The skiq_read_rx_LO_freq() function reads the current setting for the LO frequency of the specified Rx interface.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_freq*	a pointer to the variable that should be updated with the programmed frequency (in Hertz)
out	*p_actual_freq*	a pointer to the variable that should be updated with the actual tuned frequency (in Hertz)

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid RX handle specified
-ENODATA	RX LO frequency has not yet been configured

5.6.4.19 EPIQ_API int32_t skiq_write_rx_LO_freq (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *freq*)

The skiq_write_rx_LO_freq() function writes the current setting for the LO frequency of the specified Rx interface.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*freq*	the new value for the LO freq (in Hertz)

Returns

int32_t status where 0=success, anything else is an error

5.6.4.20 EPIQ_API int32_t skiq_read_rx_sample_rate (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t ∗ *p_rate*, double ∗ *p_actual_rate*)

The skiq_read_rx_sample_rate() function reads the current setting for the rate of received samples being transferred into the FPGA from the RFIC.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_rate*	a pointer to the variable that should be updated with the current sample rate setting (in Hertz) currently set for the specified interface
out	*p_actual_rate*	a pointer to the variable that should be updated with the actual rate of received samples being transferred into the FPGA

Returns

int32_t status where 0=success, anything else is an error

5.6.4.21 EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t *rate*, uint32_t *bandwidth*)

The skiq_write_rx_sample_rate_and_bandwidth() function writes the current setting for the rate of received samples being transferred into the FPGA from the RFIC. Additionally, the channel bandwidth is also configured.

Note

When configuring multiple handles, skiq_write_rx_sample_rate_and_bandwidth_multi() is preferred since it offers better performance compared to multiple calls to skiq_write_rx_sample_rate_and_bandwidth().

Warning

Rx/Tx sample rates are derived from the same clock so modifications to the Rx sample rate will also update the Tx sample rate to the same value.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_write_rx_sample_rate_and_bandwidth_multi

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*rate*	the new value of the sample rate (in Hertz)
in	*bandwidth*	specifies the channel bandwidth in Hertz

Returns

int32_t status where 0=success, anything else is an error

5.6.4.22 EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth_multi (uint8_t *card*, skiq_rx_hdl_t *handles[ ]*, uint8_t *nr_handles*, uint32_t *rate[ ]*, uint32_t *bandwidth[ ]*)

The skiq_write_rx_sample_rate_and_bandwidth_multi() function allows users to configure the sample rate and bandwith for multiple receive handles.

Note

> This function is preferred when configuring multiple handles, as it offers better performance compared to multiple calls to skiq_write_rx_sample_rate_and_bandwidth().

Warning

> Rx/Tx sample rates are derived from the same clock so modifications to the Rx sample rate will also update the Tx sample rate to the same value.

Since

> 4.15.0

See Also

> skiq_write_rx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*handles*	[skiq_rx_hdl_t] array of rx handles to be initialized
in	*nr_handles*	number of rx handles defined in `handles`
in	*rate*	array of sample rates corresponding to handles[]
in	*bandwidth*	array of bandwidth values corresponding to handles[]

Returns

> int32_t status where 0=success, anything else is an error

Return values

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENOSYS	if the FPGA version does not support IQ ordering mode
-ENOTSUP	if IQ order mode is not supported for the loaded FPGA bitstream
-EINVAL	if an invalid rate or bandwidth is specified

Note

> The indices of `handles`[] and `rate`[] should line up such that index N descibes the libsidekiq rx_handle of interest, the sample rate for index N (in `rate`[]), and the bandwidth for index N (in `bandwidth`[]) For example:

```
card = 1
handles[0] = skiq_rx_hdl_A1
handles[1] = skiq_rx_hdl_B1
```

```
rate[0] =  61440000
rate[1] = 122880000
bandwidth[0] =  49152000
bandwidth[1] = 100000000
nr_handles = 2;
```

means libsidekiq card 1 will be configured to receive on handle skiq_rx_hdl_A1 @ 61440000 Msps with a bandwidth of 49152000 Hz and skiq_rx_hdl_B1 @ 122880000 Msps with a bandwidth of 100000000 Hz.

5.6.4.23 EPIQ_API int32_t skiq_read_rx_sample_rate_and_bandwidth (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t ∗ *p_rate*, double ∗ *p_actual_rate*, uint32_t ∗ *p_bandwidth*, uint32_t ∗ *p_actual_bandwidth*)

The skiq_read_rx_sample_rate_and_bandwidth() function reads the current setting for the rate of received samples being transferred into the FPGA from the RFIC and the configured channel bandwidth.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_rate*	a pointer to the variable that should be updated with the current sample rate setting (in Hertz) currently set for the specified interface
out	*p_actual_rate*	a pointer to the variable that should be updated with the actual rate of received samples being transferred into the FPGA
out	*p_bandwidth*	a pointer to the variable that is updated with the current channel bandwidth setting (in Hertz)
out	*p_actual_-bandwidth*	a pointer to the variable that is updated with the actual channel bandwidth configured (in Hertz)

Returns

int32_t status where 0=success, anything else is an error

5.6.4.24 EPIQ_API int32_t skiq_set_rx_transfer_timeout (const uint8_t *card*, const int32_t *timeout_us*)

The skiq_set_rx_transfer_timeout() function is responsible for updating the current receive transfer timeout for the provided card. The currently permitted range of timeout is RX_TRANSFER_WAIT_FOREVER, RX_TRANSFER_NO_WAIT, or a value between 20 and 1000000.

Note

Changing the receive transfer timeout may affect calls that are in progress.
A skiq_receive() call that times out is only guaranteed to be at least the receive transfer timeout value, and makes no guarantee of an upper bound. Once the timeout has been exceeded without a packet from the FPGA, the call returns at the next opportunity the kernel provides to the associated process.

Warning

When using a non-zero timeout, calling skiq_stop_rx_streaming() or skiq_exit() can cause skiq_receive() to return without a packet. Be sure to handle that case.

Note

For improved CPU usage efficiency in receiving, a non-zero timeout is recommended. Additionally, a timeout that is greater than the inter-block timing at the configured Rx sample rate is also recommended.

See Also

skiq_receive
skiq_get_rx_transfer_timeout

Parameters

in	*card*	card index of the Sidekiq of interest
in	*timeout_us*	minimum timeout in microseconds for a blocking skiq_receive(). can be RX_TRANSFER_WAIT_FOREVER, RX_TRANSFER_NO_WAIT, or 20-1000000.

Returns

int32_t status where 0=success, anything else is an error.

5.6.4.25 EPIQ_API int32_t skiq_get_rx_transfer_timeout (const uint8_t *card*, int32_t * *p_timeout_us*)

The skiq_get_rx_transfer_timeout() function returns the currently configured receive transfer timeout. If the return code indicates success, then p_timeout_us is guaranteed to be RX_TRANSFER_NO_WAIT, RX_TRANSFER_WAIT_FOREVER, RX_TRANSFER_WAIT_NOT_SUPPORTED or 20-1000000.

See Also

skiq_receive
skiq_set_rx_transfer_timeout

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_timeout_us*	reference to an int32_t to populate

Returns

int32_t status where 0=success, anything else is an error.

5.6.4.26 EPIQ_API skiq_rx_status_t skiq_receive (uint8_t *card*, skiq_rx_hdl_t * *p_hdl*, skiq_rx_block_t ** *pp_block*, uint32_t * *p_data_len*)

The skiq_receive() function is responsible for receiving a contiguous block of data from the FPGA. The type of data being returned is specified in the metadata, but is typically timestamped I/Q samples. One contiguous block of data will be returned each time this function is called.

Warning

The Rx interface from which the data was received is specified in the p_hdl parameter. This is needed because the underlying driver may have multiple Rx interfaces streaming simultaneously, and these data streams will be interleaved by the hardware.

Attention

The format of the data returned by the receive call is specified by the skiq_rx_block_t structure.
See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_start_rx_streaming
skiq_start_rx_streaming_on_1pps
skiq_stop_rx_streaming
skiq_stop_rx_streaming_on_1pps
skiq_rx_block_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_hdl*	[skiq_rx_hdl_t] a pointer to the Rx handle that will be updated by libsidekiq to specify the handle associated with the received data
out	*pp_block*	[skiq_rx_block_t] a reference to a receive block reference
out	*p_data_len*	a pointer to be filled in with the # of bytes are returned as part of the transfer

Returns

skiq_rx_status_t status of the receive call

5.6.4.27 EPIQ_API int32_t skiq_read_rx_gain_index_range (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t ∗ *p_gain_index_min*, uint8_t ∗ *p_gain_index_max*)

The skiq_read_rx_gain_index_range() function is responsible for obtaining the viable range of gain index values that can be used to call into the skiq_write_rx_gain() function. Note that the range provided is inclusive.

Since

Function added in API **v4.2.0**

See Also

skiq_write_rx_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the desired interface
out	*p_gain_index_-min*	pointer to be updated with minimum index value
out	*p_gain_index_-max*	pointer to be updated with maximum index value

Returns

int32_t status where 0=success, anything else is an error

5.6.4.28 EPIQ_API int32_t skiq_write_rx_gain (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t *gain_index*)

The skiq_write_rx_gain() function is responsible for setting the overall gain of the Rx lineup for the specified receiver by means of providing an index that maps to a specified gain. The gain index value is a direct index into the gain table of the radio. The mapping of gain index to gain in dB is dependent on the RFIC used by the product.

- For Sidekiq mPCIe (skiq_mpcie), Sidekiq M.2 (skiq_m2), Sidekiq Stretch (skiq_m2_2280), Sidekiq Z2 (skiq_z2), and Matchstiq Z3u (skiq_z3u) each increment of the gain index value results in approximately 1 dB of gain, with approximately 76 dB of total gain available. For details on the gain table, refer to p. 37 of `AD9361 Reference Manual UG-570`
- For Sidekiq X2 (skiq_x2), the A1 (Rx1) & A2 (Rx2) receivers have approximately 30 dB of total gain available, where an increment of 1 in the gain index value results in approximately 0.5 dB increase. The B1 (ObsRx) receiver has approximately 18 dB of total gain available, where an increment of 1 in the gain index value results in approximately 1 dB increase in gain. For details on the gain table available, refer to the "Gain Table" section on p. 120 of the AD9371 User Guide (UG-992)
- For Sidekiq X4 (skiq_x4), each receiver has 30 dB of total gain available, where an increment of 1 in the gain index results in approximately 0.5 dB increase. For details on the receiver datapath and gain control blocks, refer to the "Receiver Datapath" on p. 125 of the ADRV9008-1/ADRV9008-2/ADRV9009 Hardware Reference Manual (UG-1295)
- For Sidekiq NV100 (skiq_nv100), each receiver has 34 dB of total gain available, where an increment of 1 in the gain index results in approximately 0.5 dB increase. For details on the gain table available, refer to the "Receiver Specifications" section on p. 6 of the `ADRV9002: Dual Narrow/Wideband RF Data Sheet`

See Also

skiq_read_rx_gain_index_range
skiq_read_rx_gain
skiq_read_rx_gain_mode
skiq_write_rx_gain_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the desired interface
in	*gain_index*	the requested rx gain index

Returns

int32_t status where 0=success, anything else is an error

5.6.4.29 EPIQ_API int32_t skiq_read_rx_gain (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t * *p_gain_index*)

The skiq_read_rx_gain() function is responsible for retrieving the current gain index for the specified Rx interface. The gain index value is a direct index into the gain table of the radio. The mapping of gain index to gain in dB is dependent on the RFIC used by the product.

- For Sidekiq mPCIe (skiq_mpcie), Sidekiq M.2 (skiq_m2), Sidekiq Stretch (skiq_m2_2280), Sidekiq Z2 (skiq_z2), and Matchstiq Z3u (skiq_z3u) each increment of the gain index value results in approximately 1 dB of gain, with approximately 76 dB of total gain available. For details on the gain table, refer to p. 37 of `AD9361 Reference Manual UG-570`
- For Sidekiq X2 (skiq_x2), the A1 (Rx1) & A2 (Rx2) receivers have approximately 30 dB of total gain available, where an increment of 1 in the gain index value results in approximately 0.5 dB increase. The B1 (ObsRx) receiver has approximately 18 dB of total gain available, where an increment of 1 in the gain index value results in approximately 1 dB increase in gain. For details on the gain table available, refer to the "Gain Table" section on p. 120 of the AD9371 User Guide (UG-992)
- For Sidekiq X4 (skiq_x4), each receiver has 30 dB of total gain available, where an increment of 1 in the gain index results in approximately 0.5 dB increase. For details on the receiver datapath and gain control blocks, refer to the "Receiver Datapath" on p.125 of the ADRV9008-1/ADRV9008-2/ADRV9009 Hardware Reference Manual (UG-1295)

See Also

skiq_read_rx_gain_index_range
skiq_read_rx_gain_mode
skiq_write_rx_gain
skiq_write_rx_gain_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the desired interface
out	*p_gain_index*	a pointer to be updated with current gain index

Returns

int32_t status where 0=success, anything else is an error

5.6.4.30 EPIQ_API int32_t skiq_read_rx_gain_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_gain_t * *p_gain_mode*)

The skiq_read_rx_gain_mode() function is responsible for reading the current gain mode being used by the Rx interface.

See Also

skiq_read_rx_gain_index_range
skiq_read_rx_gain
skiq_write_rx_gain
skiq_write_rx_gain_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_gain_mode*	[skiq_rx_gain_t] a pointer to where the currently set Rx gain mode will be written. Valid values are skiq_rx_gain_manual and skiq_rx_gain_auto.

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.31 EPIQ_API int32_t skiq_write_rx_gain_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_gain_t *gain_mode*)

The skiq_write_rx_gain_mode() function is responsible for writing the current gain mode being used by the Rx interface.

See Also

skiq_read_rx_gain_index_range
skiq_read_rx_gain
skiq_read_rx_gain_mode
skiq_write_rx_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested Rx interface
in	*gain_mode*	[skiq_rx_gain_t] the requested Rx gain mode to be written. Valid values are skiq_rx_gain_manual and skiq_rx_gain_auto.

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.32 EPIQ_API int32_t skiq_write_rx_attenuation_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_attenuation_mode_t *mode*)

The skiq_write_rx_attenuation_mode() function is responsible for writing the current attenuation mode being used by the Rx interface.

Attention

This is only supported for `Sidekiq X2`.

Since

Function added in API **v4.4.0**

See Also

skiq_read_rx_attenuation
skiq_read_rx_attenuation_mode
skiq_write_rx_attenuation

Parameters

card	card index of the Sidekiq of interest
hdl	the handle of the requested Rx interface
mode	[skiq_rx_attenuation_mode_t] the requested Rx attenuation mode to be written

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.33 EPIQ_API int32_t skiq_read_rx_attenuation_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_attenuation_mode_t * *p_mode*)

The skiq_read_rx_attenuation_mode() function is responsible for reading the current attenuation mode being used by the Rx interface.

Attention

This is only supported for `Sidekiq X2`.

Since

Function added in API **v4.4.0**

See Also

skiq_read_rx_attenuation
skiq_write_rx_attenuation
skiq_write_rx_attenuation_mode

Parameters

`in`	*card*	card index of the Sidekiq of interest

in	*hdl*	[skiq_rx_hdl_t] the handle of the requested Rx interface
out	*p_mode*	[skiq_rx_attenuation_mode_t] pointer to be updated with the current Rx attenuation mode

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.34 EPIQ_API int32_t skiq_write_rx_attenuation (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t *attenuation*)

The skiq_write_rx_attenuation() function is responsible for writing the Rx attenuation in 0.25 dB steps. Note that the Rx attenuation is applied to an external analog attenuator before the Rx signal reaches the RFIC.

Attention

This is only supported for `Sidekiq X2`. Refer to the `Sidekiq X2 Hardware User's Manual` for further details. This function will write the attenuators called out in "Figure 2: Sidekiq X2 block diagram". Attenuator "att2" maps to skiq_rx_hdl_A1, "att1" maps to skiq_rx_hdl_A2, and "att3" maps to skiq_rx_hdl_B1.

Since

Function added in API **v4.4.0**

See Also

skiq_read_rx_attenuation
skiq_read_rx_attenuation_mode
skiq_write_rx_attenuation_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested Rx interface
in	*attenuation*	the attenuation to be applied in quarter dB steps

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.35 EPIQ_API int32_t skiq_read_rx_attenuation (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t * *p_attenuation*)

The skiq_read_rx_attenuation() function is responsible for reading the current Rx attenuation, returned in 0.25 dB steps. Note that the Rx attenuation is read from an external analog attenuator before the Rx signal reaches the RFIC.

Attention

This is only supported for Sidekiq X2. Refer to the Sidekiq X2 Hardware User's Manual for further details. This function will write the attenuators called out in "Figure 2: Sidekiq X2 block diagram". Attenuator "att2" maps to skiq_rx_hdl_A1, "att1" maps to skiq_rx_hdl_A2, and "att3" maps to skiq_rx_hdl_B1.

Since

Function added in API **v4.4.0**

See Also

skiq_read_rx_attenuation_mode
skiq_write_rx_attenuation
skiq_write_rx_attenuation_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_attenuation*	pointer to take current attenuation in quarter dB steps

Returns

int32_t: status where 0=success, anything else is an error

5.6.4.36 EPIQ_API int32_t skiq_write_rx_stream_mode (uint8_t *card*, skiq_rx_stream_mode_t *stream_mode*)

The skiq_write_rx_stream_mode() function is responsible for setting the receive stream mode for a specified Sidekiq card. This must be set prior to calling skiq_start_rx_streaming() for any Rx interface associated with the card.

Warning

If this function is called after **any** Rx interface has started streaming, the setting will be stored but will not be used until all receive streaming has stopped and re-started for the card.

Attention

If the receive stream mode is set to skiq_rx_stream_mode_low_latency and an incompatible FPGA bitstream is then loaded via skiq_prog_fpga_from_file(), skiq_prog_fpga_from_flash() or skiq_prog_fpga_from_flash_slot(), the mode will automatically revert to skiq_rx_stream_mode_high_tput without warning.

Warning

skiq_rx_stream_mode_low_latency conflicts with I/Q pack mode. As such, caller may not configure a card to use both packed I/Q mode and RX low latency mode at the same time. This function will return an error (`-EPERM`) if caller sets stream_mode to skiq_rx_stream_mode_low_latency and I/Q pack mode is currently set to `true`.

Since

Function added in **v4.6.0**, requires FPGA **v3.9.0** or later

See Also

skiq_read_rx_stream_mode
skiq_rx_stream_mode_t

Parameters

in	*card*	card index of the Sidekiq of interest
in	*stream_mode*	[skiq_rx_stream_mode_t] the desired stream mode for the receive sample blocks

Returns

int32_t

Return values

0	successful setting of RX stream mode
-1	specified card index is out of range or has not been initialized
-ENOTSUP	specified RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	specified RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes
-EPERM	I/Q packed mode is already enabled and conflicts with the requested RX stream mode

5.6.4.37 EPIQ_API int32_t skiq_read_rx_stream_mode (uint8_t *card*, skiq_rx_stream_mode_t ∗ *p_stream_mode*)

The skiq_read_rx_stream_mode() function is responsible for retrieving the currently stored receive stream mode (skiq_rx_stream_mode_t).

Attention

The receive stream mode is only applied when receive streaming is started and thus may not reflect the current stream state.

Since

Function added in **v4.6.0**, requires FPGA **v3.9.0** or later

See Also

skiq_write_rx_stream_mode
skiq_rx_stream_mode_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_stream_mode*	[skiq_rx_stream_mode_t] the current value of the receive stream mode

Returns

int32_t

Return values

0	successful query of RX stream mode
-1	specified card index is out of range or has not been initialized

5.6.4.38 EPIQ_API int32_t skiq_write_rx_dc_offset_corr (uint8_t *card*, skiq_rx_hdl_t *hdl*, bool *enable*)

The skiq_write_rx_dc_offset_corr() function is used to configure the DC offset correction block in the FPGA. This is a simple 1-pole filter with a knee very close to DC.

See Also

skiq_read_rx_dc_offset_corr

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the Rx interface to access
in	*enable*	true to enable the DC offset correction block

Returns

int32_t status where 0=success, anything else is an error

5.6.4.39 EPIQ_API int32_t skiq_read_rx_dc_offset_corr (uint8_t *card*, skiq_rx_hdl_t *hdl*, bool * *p_enable*)

The skiq_read_rx_dc_offset_corr() function is responsible for returning whether the FPGA-based DC offset correction block is enabled.

See Also

skiq_write_rx_dc_offset_corr

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the Rx interface to access

out	*p_enable*	pointer to where to store the enable state, true indicates that DC offset correction block is enabled

Returns

int32_t status where 0=success, anything else is an error

5.6.4.40 EPIQ_API int32_t skiq_write_rx_fir_gain (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_fir_gain_t *gain*)

The skiq_write_rx_fir_gain() function is responsible for configuring the gain of the Rx FIR filter. The Rx FIR filter is used in configuring the Rx channel bandwidth.

See Also

skiq_read_rx_fir_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the Rx interface to access
in	*gain*	[skiq_rx_fir_gain_t] gain of the filter

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.6.4.41 EPIQ_API int32_t skiq_read_rx_fir_gain (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_fir_gain_t ∗ *p_gain*)

The skiq_read_rx_fir_gain() function is responsible for reading the gain of the Rx FIR filter. The Rx FIR filter is used in configuring the Rx channel bandwidth.

See Also

skiq_write_rx_fir_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the Rx interface to access
out	*p_gain*	[skiq_rx_fir_gain_t] pointer to where to store the gain setting

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.6.4.42 EPIQ_API int32_t skiq_read_num_rx_chans (uint8_t *card*, uint8_t * *p_num_rx_chans*)

The skiq_read_num_rx_chans() function is responsible for returning the number of Rx channels supported for the Sidekiq card of interest. The handle for the first Rx interface is skiq_rx_hdl_A1 and increments from there.

See Also

skiq_rx_hdl_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_num_rx_chans*	pointer to the number of Rx channels

Returns

int32_t status where 0=success, anything else is an error

5.6.4.43 EPIQ_API int32_t skiq_read_rx_iq_resolution (uint8_t *card*, uint8_t * *p_adc_res*)

The skiq_read_rx_iq_resolution() function is responsible for returning the resolution (in bits) per RX (ADC) IQ sample.

Since

Function added in API **v4.2.0**

See Also

skiq_receive
skiq_rx_block_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_adc_res*	pointer to where to store the ADC resolution

Returns

int32_t status where 0=success, anything else is an error

5.6.4.44 EPIQ_API int32_t skiq_read_rx_LO_freq_range (uint8_t *card*, uint64_t * *p_max*, uint64_t * *p_min*)

The skiq_read_rx_LO_freq_range() function allows an application to obtain the maximum and minimum LO frequencies that a Sidekiq can tune to receive. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_max_rx_LO_freq
skiq_read_min_rx_LO_freq
skiq_read_parameters
skiq_rx_param_t::lo_freq_min
skiq_rx_param_t::lo_freq_max

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_max*	pointer to update with maximum LO frequency
out	*p_min*	pointer to update with minimum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.6.4.45 EPIQ_API int32_t skiq_read_max_rx_LO_freq (uint8_t *card*, uint64_t * *p_max*)

The skiq_read_max_rx_LO_freq() function allows an application to obtain the maximum LO frequency that a Sidekiq can tune to receive. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_rx_LO_freq_range
skiq_read_min_rx_LO_freq
skiq_read_parameters
skiq_rx_param_t::lo_freq_max

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_max*	pointer to update with maximum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.6.4.46 EPIQ_API int32_t skiq_read_min_rx_LO_freq (uint8_t *card*, uint64_t * *p_min*)

The skiq_read_min_rx_LO_freq() function allows an application to obtain minimum LO frequency that a Sidekiq can tune to receive. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_rx_LO_freq_range
skiq_read_max_rx_LO_freq
skiq_read_parameters
skiq_rx_param_t::lo_freq_min

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_min*	pointer to update with minimum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.6.4.47 EPIQ_API int32_t skiq_read_rx_block_size (uint8_t *card*, skiq_rx_stream_mode_t *stream_mode*)

The skiq_read_rx_block_size returns the expected RX block size (in bytes) for a specified skiq_rx_stream_mode_t.

Since

Function added in API **v4.6.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*stream_mode*	[skiq_rx_stream_mode_t] RX stream mode associated with RX block size

Returns

int32_t

Return values

>0	expected block size (in bytes) for the specified RX stream mode
-1	specified card index is out of range or has not been initialized
-ENOTSUP	specified RX stream mode is not supported for the loaded FPGA bitstream
-EINVAL	specified RX stream mode is not a valid mode, see skiq_rx_stream_mode_t for valid modes

5.6.4.48 EPIQ_API int32_t skiq_read_rx_cal_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_cal_mode_t ∗ *p_mode*)

The skiq_read_rx_cal_mode() function reads the RX calibration mode.

Since

Function added in API **v4.13.0**

See Also

skiq_write_rx_cal_mode
skiq_run_rx_cal
skiq_read_rx_cal_type_mask
skiq_write_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_mode*	[skiq_rx_cal_mode_t] the currently set value of the RX calibration mode setting

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EFAULT	NULL pointer detected for `p_mode`

5.6.4.49 EPIQ_API int32_t skiq_write_rx_cal_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rx_cal_mode_t *mode*)

The skiq_write_rx_cal_mode() function writes the RX calibration mode. If automatic mode is configured, writing the RX LO frequency may result in the RX calibrations to be performed prior to completing the tune operation. The types of calibrations performed are controlled by the [skiq_rx_cal_type_t] configuration. If manual mode is configured, it is the user's responsibility to determine when to run the RX calibration via skiq_run_rx_cal().

Since

Function added in API **v4.13.0**

See Also

skiq_read_rx_cal_mode
skiq_run_rx_cal
skiq_read_rx_cal_type_mask
skiq_write_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*mode*	[skiq_rx_cal_mode_t] RX calibration mode to configure

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized

-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.6.4.50 EPIQ_API int32_t skiq_run_rx_cal (uint8_t *card*, skiq_rx_hdl_t *hdl*)

The skiq_run_rx_cal() performs the RX calibration based on the current RFIC settings and RX calibrations enabled.

Note

that this may take some time to complete, depending on the calibration types enabled, RF environment, the Sidekiq product (<100 ms to >1 second).
streaming RX or TX while running the RX calibration will result in a momentary gap in received and/or transmitted samples. It is recommended that the function is ran after the desired RX LO frequency has been configured.

Attention

In the case of Sidekiq X4, calibration is performed on all enabled RX handles, regardless of the handle specified.

Since

Function added in API **v4.13.0**

See Also

skiq_read_rx_cal_mode
skiq_write_rx_cal_mode
skiq_read_rx_cal_type_mask
skiq_write_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENODEV	Generic error accessing card
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.6.4.51 EPIQ_API int32_t skiq_read_rx_cal_type_mask (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t * *p_cal_mask*)

The skiq_read_rx_cal_type_mask() function reads the RX calibration types configured.

Since

Function added in API **v4.13.0**

See Also

skiq_read_rx_cal_mode
skiq_write_rx_cal_mode
skiq_run_rx_cal
skiq_write_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_cal_mask*	a bitmask of the currently enabled RX calibration types

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EFAULT	NULL pointer detected for `p_cal_mask`

5.6.4.52 EPIQ_API int32_t skiq_write_rx_cal_type_mask (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t *cal_mask*)

The skiq_write_rx_cal_type_mask() function writes the RX calibration types to use when calibration is ran either manuallly or automatically.

Since

Function added in API **v4.13.0**

See Also

skiq_read_rx_cal_mode
skiq_write_rx_cal_mode
skiq_run_rx_cal
skiq_read_rx_cal_type_mask
skiq_read_rx_cal_types_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*cal_mask*	bitmask of calibration types to perform. This should be formed by ORing [skiq_rx_cal_type_t] for each calibration type to enable.

Returns

int32_t status where 0=success, else a negative errno value

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EINVAL	Invalid mask specified for product

5.6.4.53 EPIQ_API int32_t skiq_read_rx_cal_types_avail (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t ∗ *p_cal_mask*)

The skiq_read_rx_cal_types_avail() function provides a bitmask of all of the RX calibration types available.

Since

Function added in API **v4.13.0**

See Also

skiq_read_rx_cal_mode
skiq_write_rx_cal_mode
skiq_run_rx_cal
skiq_read_rx_cal_type_mask
skiq_write_rx_cal_type_mask

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_cal_mask*	pointer to a bitmask of the RX calibration types [skiq_rx_cal_type_t] available

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EFAULT	NULL pointer detected for `p_cal_mask`

5.6.4.54 EPIQ_API int32_t skiq_read_rx_analog_filter_bandwidth (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t ∗ *p_bandwidth*)

The skiq_read_rx_analog_filter_bandwidth() function reads the current setting for the RX analog filter bandwidth.

Since

Function added in 4.17.0

Note

that this value is automatically updated when the channel bandwidth is changed
This is not available for all products

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_bandwidth*	pointer to the variable that should be updated with the actual bandwidth of the analog filter bandwidth

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.6.4.55 EPIQ_API int32_t skiq_write_rx_analog_filter_bandwidth (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint32_t *bandwidth*)

The skiq_write_rx_analog_filter_bandwidth() function writes the current bandwidth of the analog filter.

Since

Function added in 4.17.0

Note

that this value is overwritten when the bandwidth is configured with skiq_write_rx_sample_rate_and_-bandwidth
This is not available for all products
not all bandwidth settings are valid and actual setting can be queried
For AD9361 products, the analog filter bandwidth is typically set to the configured channel bandwidth and is automatically configured to this value when the sample rate and channel bandwidth is configured. This function allows the analog filter bandwidth to be overwritten, where the corner frequency of the 3rd order Butterworth filter is set to 1.4x of half the specified bandwidth.

See Also

skiq_write_rx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*bandwidth*	specifies the analog filter bandwidth in Hertz

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.6.5 Variable Documentation

5.6.5.1 EPIQ_API const char* SKIQ_RX_STREAM_MODE_STRINGS[skiq_rx_stream_mode_end]

String representation of skiq_rx_stream_mode_t.

See Also

skiq_rx_stream_mode_t

Definition at line 491 of file sidekiq_types.h.

5.7 Transmit Functions and Definitions

These functions and definitions are related to configuring and exercising the transmit capabilities of the Sidekiq SDR.

Classes

- struct skiq_tx_block_t

  Sidekiq Transmit Block type definition for use with skiq_transmit and skiq_tx_callback_t.

Macros

- #define SKIQ_MAX_TX_PACKET_SIZE_IN_WORDS 65536

  The largest number of words that can be transferred between the FPGA and the CPU. This includes both the data block as well as the header size.
- #define SKIQ_TX_HEADER_SIZE_IN_WORDS 4

  The current Tx header size is fixed at 4 words of metadata for now at the start of each I/Q block, which may well increase at some point. For details on the exact format and contents of the transmit packet, refer to skiq_transmit()
- #define SKIQ_TX_TIMESTAMP_OFFSET_IN_WORDS 2

  The offset (in 32-bit words) to the header where the Tx timestamp is stored.
- #define SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS

  SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS is the largest block size of sample data that can be transferred from the CPU to the FPGA while transmitting. Note that a "block" of data includes the sample data minus the header data
- #define SKIQ_TX_HEADER_SIZE_IN_BYTES

  The current Tx header size, only in bytes.
- #define SKIQ_TX_PACKET_SIZE_INCREMENT_IN_WORDS (256)

  The Tx packet must be in increments of 256 words. Note: the packet size accounts for both the header size as well as the block (sample) size.
- #define SKIQ_TX_ASYNC_SEND_QUEUE_FULL (100)

  SKIQ_TX_ASYNC_SEND_QUEUE_FULL is the return code of the skiq_transmit() call when using skiq_tx_transfer_mode_async and there is no space available to store the data to send
- #define SKIQ_TX_MAX_NUM_THREADS (10)

  SKIQ_TX_MAX_NUM_THREADS is the maximum number of threads used in transmitting when using skiq_tx_transfer_mode_async
- #define SKIQ_TX_MIN_NUM_THREADS (2)

  SKIQ_TX_MIN_NUM_THREADS is the minimum number of threads used in transmitting when skiq_tx_transfer_mode_async
- #define SKIQ_MAX_NUM_TX_QUEUED_PACKETS (50)

  Maximum number of TX packets that can be queued when running in skiq_tx_transfer_mode_async.
- #define SKIQ_MAX_NUM_FREQ_HOPS (512)

  Maximum number of frequencies that can be specified in a hopping list.
- #define SKIQ_TX_BLOCK_MEMORY_ALIGN 4096

  Defines the memory alignment of a transmit block when allocated using SKIQ_TX_BLOCK_INITIALIZER, SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS, SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES, skiq_tx_block_allocate() or skiq_tx_block_allocate_by_bytes()
- #define SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES(var_name, data_size_in_bytes)
- #define SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS(var_name, data_size_in_words)
- #define SKIQ_TX_BLOCK_INITIALIZER(var_name)

Typedefs

- typedef void(∗ skiq_tx_callback_t)(int32_t status, skiq_tx_block_t ∗p_block, void ∗p_user)

  Transmit callback function type definition.
- typedef void(∗ skiq_tx_ena_callback_t)(uint8_t card, int32_t status)

  Transmit enabled callback function type definition where card is the Sidekiq card whose transmitter has been enabled and status is the error code associated with enabling the transmitter (0 is success)

Enumerations

- enum skiq_tx_timestamp_base_t { skiq_tx_rf_timestamp =0, skiq_tx_system_timestamp }

  Tx supports transmitting on System Timestamp or RF Timestamp on certain Sidekiq products.
- enum skiq_tx_flow_mode_t { skiq_tx_immediate_data_flow_mode =0, skiq_tx_with_timestamps_data_flow_mode, skiq_tx_with_timestamps_allow_late_data_flow_mode }

  There are several different data flow modes that can be used when transmitting data on a Sidekiq Tx interface:
- enum skiq_tx_transfer_mode_t { skiq_tx_transfer_mode_sync =0, skiq_tx_transfer_mode_async }

  There are different transfer modes that can be used when transmitting data:
- enum skiq_tx_hdl_t {
  skiq_tx_hdl_A1 =0, skiq_tx_hdl_A2 =1, skiq_tx_hdl_B1 =2, skiq_tx_hdl_B2 =3,
  skiq_tx_hdl_end }

  Sidekiq supports a single Tx interface handles. The skiq_tx_hdl_t enum is used to define the Tx interface handle.
- enum skiq_tx_fir_gain_t { skiq_tx_fir_gain_neg_6 =1, skiq_tx_fir_gain_0 =0 }

  Tx FIR filter gain settings, applied to the Tx FIR used in the Tx channel bandwidth configuration.
- enum skiq_tx_quadcal_mode_t { skiq_tx_quadcal_mode_auto =0, skiq_tx_quadcal_mode_manual }

  TX Quadrature Calibration Mode.

Functions

- static void skiq_tx_set_block_timestamp (skiq_tx_block_t ∗p_block, uint64_t timestamp)
- static uint64_t skiq_tx_get_block_timestamp (skiq_tx_block_t ∗p_block)
- static skiq_tx_block_t ∗ skiq_tx_block_allocate_by_bytes (uint32_t data_size_in_bytes)
- static skiq_tx_block_t ∗ skiq_tx_block_allocate (uint32_t data_size_in_samples)
- static void skiq_tx_block_free (skiq_tx_block_t ∗p_block)
- EPIQ_API int32_t skiq_start_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_start_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_stop_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_stop_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_read_tx_timestamp_base (uint8_t card, skiq_tx_timestamp_base_t ∗p_timestamp_base)
- EPIQ_API int32_t skiq_write_tx_timestamp_base (uint8_t card, skiq_tx_timestamp_base_t timestamp_base)
- EPIQ_API int32_t skiq_read_tx_data_flow_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_flow_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_tx_data_flow_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_flow_mode_t mode)
- EPIQ_API int32_t skiq_read_tx_transfer_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_transfer_mode_t ∗p_transfer_mode)

- EPIQ_API int32_t skiq_write_tx_transfer_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_transfer_mode_t transfer_mode)
- EPIQ_API int32_t skiq_register_tx_complete_callback (uint8_t card, skiq_tx_callback_t tx_complete)
- EPIQ_API int32_t skiq_register_tx_enabled_callback (uint8_t card, skiq_tx_ena_callback_t tx_ena_cb)
- EPIQ_API int32_t skiq_write_tx_filter_path (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t path)
- EPIQ_API int32_t skiq_read_tx_filter_path (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t ∗p_path)
- EPIQ_API int32_t skiq_write_tx_sample_rate_and_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t rate, uint32_t bandwidth)
- EPIQ_API int32_t skiq_read_tx_sample_rate_and_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t ∗p_rate, double ∗p_actual_rate, uint32_t ∗p_bandwidth, uint32_t ∗p_actual_bandwidth)
- EPIQ_API int32_t skiq_transmit (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_block_t ∗p_block, void ∗p_user)
- EPIQ_API int32_t skiq_read_tx_LO_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t ∗p_freq, double ∗p_tuned_freq)
- EPIQ_API int32_t skiq_write_tx_LO_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t freq)
- EPIQ_API int32_t skiq_enable_tx_tone (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_disable_tx_tone (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_tx_tone_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t ∗p_freq)
- EPIQ_API int32_t skiq_read_tx_tone_freq_offset (uint8_t card, skiq_tx_hdl_t hdl, int32_t ∗p_freq_offset)
- EPIQ_API int32_t skiq_write_tx_tone_freq_offset (uint8_t card, skiq_tx_hdl_t hdl, int32_t test_freq_offset)
- EPIQ_API int32_t skiq_write_tx_attenuation (uint8_t card, skiq_tx_hdl_t hdl, uint16_t attenuation)
- EPIQ_API int32_t skiq_read_tx_attenuation (uint8_t card, skiq_tx_hdl_t hdl, uint16_t ∗p_attenuation)
- EPIQ_API int32_t skiq_read_tx_sample_rate (uint8_t card, skiq_tx_hdl_t hdl, uint32_t ∗p_rate, double ∗p_actual_rate)
- EPIQ_API int32_t skiq_read_tx_block_size (uint8_t card, skiq_tx_hdl_t hdl, uint16_t ∗p_block_size_in_words)
- EPIQ_API int32_t skiq_write_tx_block_size (uint8_t card, skiq_tx_hdl_t hdl, uint16_t block_size_in_words)
- EPIQ_API int32_t skiq_read_tx_num_underruns (uint8_t card, skiq_tx_hdl_t hdl, uint32_t ∗p_num_underrun)
- EPIQ_API int32_t skiq_read_tx_num_late_timestamps (uint8_t card, skiq_tx_hdl_t hdl, uint32_t ∗p_num_late)
- EPIQ_API int32_t skiq_write_tx_fir_gain (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_fir_gain_t gain)
- EPIQ_API int32_t skiq_read_tx_fir_gain (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_fir_gain_t ∗p_gain)
- EPIQ_API int32_t skiq_read_num_tx_threads (uint8_t card, uint8_t ∗p_num_threads)
- EPIQ_API int32_t skiq_write_num_tx_threads (uint8_t card, uint8_t num_threads)
- EPIQ_API int32_t skiq_read_tx_thread_priority (uint8_t card, int32_t ∗p_priority)
- EPIQ_API int32_t skiq_write_tx_thread_priority (uint8_t card, int32_t priority)
- EPIQ_API int32_t skiq_read_num_tx_chans (uint8_t card, uint8_t ∗p_num_tx_chans)
- EPIQ_API int32_t skiq_read_tx_iq_resolution (uint8_t card, uint8_t ∗p_dac_res)
- EPIQ_API int32_t skiq_read_tx_LO_freq_range (uint8_t card, uint64_t ∗p_max, uint64_t ∗p_min)
- EPIQ_API int32_t skiq_read_max_tx_LO_freq (uint8_t card, uint64_t ∗p_max)
- EPIQ_API int32_t skiq_read_min_tx_LO_freq (uint8_t card, uint64_t ∗p_min)
- EPIQ_API int32_t skiq_read_tx_quadcal_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_quadcal_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_tx_quadcal_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_quadcal_mode_t mode)
- EPIQ_API int32_t skiq_run_tx_quadcal (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_tx_analog_filter_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t ∗p_bandwidth)
- EPIQ_API int32_t skiq_write_tx_analog_filter_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t bandwidth)

5.7.1 Detailed Description

These functions and definitions are related to configuring and exercising the transmit capabilities of the Sidekiq SDR.

5.7.2 Macro Definition Documentation

5.7.2.1 #define SKIQ_MAX_TX_PACKET_SIZE_IN_WORDS 65536

The largest number of words that can be transferred between the FPGA and the CPU. This includes both the data block as well as the header size.

Definition at line 416 of file sidekiq_api.h.

5.7.2.2 #define SKIQ_TX_HEADER_SIZE_IN_WORDS 4

The current Tx header size is fixed at 4 words of metadata for now at the start of each I/Q block, which may well increase at some point. For details on the exact format and contents of the transmit packet, refer to skiq_transmit()

Definition at line 422 of file sidekiq_api.h.

5.7.2.3 #define SKIQ_TX_TIMESTAMP_OFFSET_IN_WORDS 2

The offset (in 32-bit words) to the header where the Tx timestamp is stored.

Definition at line 426 of file sidekiq_api.h.

5.7.2.4 #define SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS

SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS is the largest block size of sample data that can be transferred from the CPU to the FPGA while transmitting. Note that a "block" of data includes the sample data minus the header data

Definition at line 432 of file sidekiq_api.h.

5.7.2.5 #define SKIQ_TX_HEADER_SIZE_IN_BYTES

The current Tx header size, only in bytes.

Definition at line 435 of file sidekiq_api.h.

5.7.2.6 #define SKIQ_TX_PACKET_SIZE_INCREMENT_IN_WORDS (256)

The Tx packet must be in increments of 256 words. Note: the packet size accounts for both the header size as well as the block (sample) size.

Definition at line 439 of file sidekiq_api.h.

5.7.2.7 #define SKIQ_TX_ASYNC_SEND_QUEUE_FULL (100)

SKIQ_TX_ASYNC_SEND_QUEUE_FULL is the return code of the skiq_transmit() call when using skiq_tx_-transfer_mode_async and there is no space available to store the data to send

Definition at line 576 of file sidekiq_api.h.

5.7.2.8 #define SKIQ_TX_MAX_NUM_THREADS (10)

SKIQ_TX_MAX_NUM_THREADS is the maximum number of threads used in transmitting when using skiq_-tx_transfer_mode_async

Definition at line 580 of file sidekiq_api.h.

5.7.2.9 #define SKIQ_TX_MIN_NUM_THREADS (2)

SKIQ_TX_MIN_NUM_THREADS is the minimum number of threads used in transmitting when skiq_tx_-transfer_mode_async

Definition at line 584 of file sidekiq_api.h.

5.7.2.10 #define SKIQ_MAX_NUM_TX_QUEUED_PACKETS (50)

Maximum number of TX packets that can be queued when running in skiq_tx_transfer_mode_async.

Definition at line 98 of file sidekiq_types.h.

5.7.2.11 #define SKIQ_MAX_NUM_FREQ_HOPS (512)

Maximum number of frequencies that can be specified in a hopping list.

Definition at line 104 of file sidekiq_types.h.

5.7.2.12 #define SKIQ_TX_BLOCK_MEMORY_ALIGN 4096

Defines the memory alignment of a transmit block when allocated using SKIQ_TX_BLOCK_INITIALIZE-R, SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS, SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES, skiq_tx_block_-allocate() or skiq_tx_block_allocate_by_bytes()

Definition at line 114 of file sidekiq_types.h.

5.7.2.13 #define SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES(*var_name*, *data_size_in_bytes*)

Sidekiq Transmit Block static initializer, user specifies the number of desired bytes.

Note

> Sidekiq Transmit Blocks statically allocated should be typecast to a skiq_tx_block_t reference when calling skiq_transmit to avoid compiler warnings

Since

> MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name
in	*data_size_in_bytes*	desired payload size (bytes)

Definition at line 1013 of file sidekiq_types.h.

5.7.2.14 #define SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS(*var_name, data_size_in_words*)

Sidekiq Transmit Block static initializer, user specifies the number of desired words

Note

Sidekiq Transmit Blocks statically allocated should be typecast to a skiq_tx_block_t reference when calling skiq_transmit to avoid compiler warnings

Since

MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name
in	*data_size_in_words*	desired payload size (words)

Definition at line 1032 of file sidekiq_types.h.

5.7.2.15 #define SKIQ_TX_BLOCK_INITIALIZER(*var_name*)

Sidekiq Transmit Block static initializer, allocates the maximum transmit block size

Note

Sidekiq Transmit Blocks statically allocated should be typecast to a skiq_tx_block_t reference when calling skiq_transmit to avoid compiler warnings

Since

MACRO added in **v4.0.0**

Parameters

in	*var_name*	desired variable name

Definition at line 1050 of file sidekiq_types.h.

5.7.3 Typedef Documentation

5.7.3.1 typedef void(∗ skiq_tx_callback_t)(int32_t status, skiq_tx_block_t ∗p_block, void ∗p_user)

Transmit callback function type definition.

See Also

skiq_register_tx_complete_callback

Definition at line 1101 of file sidekiq_types.h.

5.7.3.2 typedef void(∗ skiq_tx_ena_callback_t)(uint8_t card, int32_t status)

Transmit enabled callback function type definition where card is the Sidekiq card whose transmitter has been enabled and status is the error code associated with enabling the transmitter (0 is success)

See Also

skiq_register_tx_enabled_callback

Definition at line 1111 of file sidekiq_types.h.

5.7.4 Enumeration Type Documentation

5.7.4.1 enum skiq_tx_timestamp_base_t

Tx supports transmitting on System Timestamp or RF Timestamp on certain Sidekiq products.

Configuration of which timestamp should be used will generally be needed for applications where timing of the transmission is a critical factor.

Call skiq_read_tx_timestamp_base() after enabling/initializing a card to determine what the default value for the card is.

Warning

When skiq_tx_rf_timestamp_base is configured the transmission of data will not occur until the next clock of the RF timestamp after the system timestamp occured. This can make a significant impact on transmit timing.

Since

Function added in API **v4.16.0**

See Also

skiq_read_tx_timestamp_base
skiq_write_tx_timestamp_base

Enumerator

skiq_tx_rf_timestamp The FPGA design compares a skiq_tx_block_t's transmit timestamp to the transmit **sample** counter which typically increments at the transmit sample rate.
See Also

skiq_read_curr_tx_timestamp

skiq_tx_system_timestamp The FPGA design compares a skiq_tx_block_t's transmit timestamp to the transmit **system** counter increments at the system clock frequency.
See Also

skiq_read_curr_sys_timestamp

Definition at line 140 of file sidekiq_types.h.

5.7.4.2 enum skiq_tx_flow_mode_t

There are several different data flow modes that can be used when transmitting data on a Sidekiq Tx interface:

- tx immediate - I/Q data is transmitted as soon as possible, without regard to timestamps.
- tx with timestamps - I/Q data is queued up by software and/or the FPGA, and only transmitted out when the appropriate timestamp has occurred.
- tx allow late timestamps - this is similar to "tx with timestamps" mode, though data with timestamps that have already passed will still be transmitted.

Tx immediate mode is generally used for applications where a transmission isn't synchronized to any other time-critical signal, and just needs to be sent out as soon as possible. Note that each packet of Tx data transferred to libsidekiq is still queued up in a FIFO, so the order of transmission is still preserved though there is no reliance on a timestamp to drive any transmission. It simply happens as quickly as possible.

Tx with timestamps mode is generally used for applications where the timing of the transmission is critical (such as in a TDMA protocol).

See Also

skiq_read_tx_data_flow_mode
skiq_write_tx_data_flow_mode

Enumerator

skiq_tx_immediate_data_flow_mode I/Q data is transmitted as soon as possible, without regard to timestamps.

skiq_tx_with_timestamps_data_flow_mode I/Q data is queued up by software and/or the FPGA, and only transmitted out when the appropriate timestamp has occurred. Data with a timestamp that already passed (late) at the time of transmit will be discarded.

skiq_tx_with_timestamps_allow_late_data_flow_mode I/Q data is queued up by software and/or the FPGA, and only transmitted out when the appropriate timestamp has occurred. Data with a timestamp that already passed (late) at the time of transmit will be transmitted.

Definition at line 190 of file sidekiq_types.h.

5.7.4.3 enum skiq_tx_transfer_mode_t

There are different transfer modes that can be used when transmitting data:

Note

For improved efficieny in transmitting, the skiq_tx_transfer_mode_async is recommended.

Attention

skiq_tx_transfer_mode_async may not be available on all Sidekiq products, check with the latest release to confirm functionality. If there are any questions, please feel free to reach out on the Epiq support forum.

See Also

skiq_read_tx_transfer_mode
skiq_write_tx_transfer_mode

Enumerator

skiq_tx_transfer_mode_sync This mode transfers packets to the FPGA synchronously. In this mode, the skiq_transmit function will block until the FPGA has received the packet of data. The FPGA FIFO for Tx packets is relatively small (see skiq_fpga_tx_fifo_size_t), so when the FIFO is full, the skiq_transmit call will block until the packet is transmitted. The length of time to actually transmit the packet depends on the sample rate.

skiq_tx_transfer_mode_async This mode transfers packets to the FPGA asynchronously. In this mode, the skiq_transmit function will schedule the packet to be transferred as long as there is enough room in the buffer for the packet. If there is not enough room to store the packet, the skiq_transmit function will return immediately with an result of SKIQ_TX_ASYNC_SEND_QUEUE_FULL. In order to run in this mode, the OS must support the ablility to schedule real-time threads and lock those threads to a specific core. When running in this mode, a callback function can be registered with the skiq_register_tx_complete_callback, which will be called once the packet transfer to the FPGA has been completed.

Definition at line 228 of file sidekiq_types.h.

5.7.4.4 enum skiq_tx_hdl_t

Sidekiq supports a single Tx interface handles. The skiq_tx_hdl_t enum is used to define the Tx interface handle.

Enumerator

skiq_tx_hdl_A1

skiq_tx_hdl_A2

skiq_tx_hdl_B1

skiq_tx_hdl_B2

skiq_tx_hdl_end

Definition at line 389 of file sidekiq_types.h.

5.7.4.5 enum skiq_tx_fir_gain_t

Tx FIR filter gain settings, applied to the Tx FIR used in the Tx channel bandwidth configuration.

See Also

skiq_read_tx_fir_gain
skiq_write_tx_fir_gain

Enumerator

skiq_tx_fir_gain_neg_6 designates a Tx FIR gain of -6 dB

skiq_tx_fir_gain_0 designates a Tx FIR gain of 0 dB

Definition at line 690 of file sidekiq_types.h.

5.7.4.6 enum skiq_tx_quadcal_mode_t

TX Quadrature Calibration Mode.

See Also

skiq_read_tx_quadcal_mode
skiq_write_tx_quadcal_mode
skiq_run_tx_quadcal

Enumerator

skiq_tx_quadcal_mode_auto automatically run TX quadrature algorithm

skiq_tx_quadcal_mode_manual do not automatically run the TX quadrature algorithm

Definition at line 819 of file sidekiq_types.h.

5.7.5 Function Documentation

5.7.5.1 static void skiq_tx_set_block_timestamp (skiq_tx_block_t ∗ *p_block*, uint64_t *timestamp*) [inline], [static]

The skiq_tx_set_block_timestamp() function sets the timestamp field (skiq_tx_block_t::timestamp) of a transmit block.

Since

Function added in v4.0.0

Parameters

in	*p_block*	[skiq_tx_block_t ∗] reference to a skiq_tx_block_t
in	*timestamp*	desired timestamp for the transmit block

Returns

void

Definition at line 664 of file sidekiq_api.h.

5.7.5.2 static uint64_t skiq_tx_get_block_timestamp (skiq_tx_block_t ∗ *p_block*) [inline], [static]

The skiq_tx_get_block_timestamp() function return the timestamp field (skiq_tx_block_t::timestamp) of a referenced transmit block.

Since

Function added in v4.0.0

Parameters

in	*p_block*	[skiq_tx_block_t *] reference to a skiq_tx_block_t

Returns

uint64_t the timestamp associated with the transmit block

Definition at line 687 of file sidekiq_api.h.

5.7.5.3 static skiq_tx_block_t* skiq_tx_block_allocate_by_bytes (uint32_t *data_size_in_bytes*) [inline], [static]

The skiq_tx_block_allocate_by_bytes() function allocates a Sidekiq Transmit Block (skiq_tx_block_t) with the desired number of bytes.

Note

The returned reference **MUST** be freed by calling skiq_tx_block_free.

Since

Function added in v4.0.0

Parameters

in	*data_size_in_bytes*	desired number of bytes in the transmit block

Returns

skiq_tx_block_t* a reference to the Sidekiq Transmit Block

Definition at line 713 of file sidekiq_api.h.

5.7.5.4 static skiq_tx_block_t* skiq_tx_block_allocate (uint32_t *data_size_in_samples*) [inline], [static]

The skiq_tx_block_allocate() function allocates a Sidekiq Transmit Block (skiq_tx_block_t) with the desired number of unpacked samples (words).

Note

The returned reference **MUST** be freed by calling skiq_tx_block_free.

Since

Function added in v4.0.0

Parameters

in	*data_size_in_-samples*	desired number of samples in the transmit block

Returns

skiq_tx_block_t* a reference to the Sidekiq Transmit Block

Definition at line 752 of file sidekiq_api.h.

5.7.5.5 static void skiq_tx_block_free (skiq_tx_block_t * *p_block*) [inline], [static]

The skiq_tx_block_free() function frees a Sidekiq Transmit Block (skiq_tx_block_t) that was allocated using skiq_tx_block_allocate().

Note

The passed reference **MUST** have been allocated by calling skiq_tx_block_allocate.

Since

Function added in v4.0.0

Parameters

in	*p_block*	[skiq_tx_block_t *] reference to the Sidekiq Transmit Block to free

Returns

void

Definition at line 773 of file sidekiq_api.h.

5.7.5.6 EPIQ_API int32_t skiq_start_tx_streaming (uint8_t *card*, skiq_tx_hdl_t *hdl*)

The skiq_start_tx_streaming() function is responsible for preparing for the flow of data between the CPU and the FPGA. Once started, the data flow can be stopped with a call to skiq_stop_tx_streaming().

The total size of the transmit packet must be in an increment of SKIQ_TX_PACKET_SIZE_INCREMENT_IN_-WORDS. The packet size is calculated by: block_size + header_size. If this condition is not met, an error will be returned and the transmit stream will not begin.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the tx interface to start streaming

Returns

int32_t status where 0=success, anything else is an error

5.7.5.7 EPIQ_API int32_t skiq_start_tx_streaming_on_1pps (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t *sys_timestamp*)

The skiq_start_tx_streaming_on_1pps() function is identical to the skiq_start_tx_streaming() with exception of when the data stream starts to flow. When calling this function, the data does not begin to flow until the rising 1PPS edge after the system timestamp specified has occurred. If a timestamp of 0 is provided, then the next 1PPS edge will begin the data flow. This function blocks until the data starts flowing.

The total size of the transmit packet must be in an increment of SKIQ_TX_PACKET_SIZE_INCREMENT_IN_WORDS. The packet size is calculated by: block_size + header_size. If this condition is not met, an error will be returned and the transmit stream will not begin.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*sys_timestamp*	system timestamp after the next 1PPS will begin the data flow

Returns

int32_t status where 0=success, anything else is an error

5.7.5.8 EPIQ_API int32_t skiq_stop_tx_streaming (uint8_t *card*, skiq_tx_hdl_t *hdl*)

The skiq_stop_tx_streaming() function is responsible for stopping the streaming of data between the CPU and the FPGA. This function can only be called after an interface has previously started streaming.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested tx interface

Returns

int32_t status where 0=success, anything else is an error

5.7.5.9 EPIQ_API int32_t skiq_stop_tx_streaming_on_1pps (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t *sys_timestamp*)

The skiq_stop_tx_streaming_on_1pps() function is identical to the skiq_stop_tx_streaming() function with the exception of when the data stops streaming. When calling this function, the data stream is disabled on the rising 1PPS edge after the system timestamp specified has occurred. If a timestamp of 0 is provided, then the next 1PPS edge will stop the data flow. This function blocks until the data flow is disabled.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested tx interface
in	*sys_timestamp*	specifies the timestamp on which to stop TX streaming

Returns

int32_t status where 0=success, anything else is an error

5.7.5.10 EPIQ_API int32_t skiq_read_tx_timestamp_base (uint8_t *card*, skiq_tx_timestamp_base_t * *p_timestamp_base*)

The skiq_read_tx_timestamp_base() function is responsible for returning the current timestamp base for transmitting on timestamp.

See Also

skiq_tx_timestamp_base_t
skiq_write_tx_timestamp_base

Since

Function added in API **v4.16.0**

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_timestamp_base*	[skiq_tx_timestamp_base_t] a pointer to the current timestamp base configuration

Returns

0 on success, else a negative errno value

Return values

-ENOSYS	if the FPGA version does not meet minimum requirements to support this feature.
-EFAULT	NULL pointer detected for `p_timestamp_base`

5.7.5.11 EPIQ_API int32_t skiq_write_tx_timestamp_base (uint8_t *card*, skiq_tx_timestamp_base_t *timestamp_base*)

The skiq_write_tx_timestamp_base() function is responsible for configuring the timestamp base for transmitting on timestamp.

See Also

skiq_tx_timestamp_base_t
skiq_read_tx_timestamp_base

Since

Function added in API **v4.16.0**

Note

This functionality is not supported on older Sidekiq mPCIe products, please contact the support forum if you have any questions about supported products.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*timestamp_base*	[skiq_tx_timestamp_base_t] timestamp base configuration desired

Returns

0 on success, else a negative errno value

Return values

-ENOTSUP	if the Sidekiq card does not support changing the base.
-ENOSYS	if the FPGA version does not meet minimum requirements to support this feature.
-EFAULT	NULL pointer detected for `p_timestamp_base`

5.7.5.12 EPIQ_API int32_t skiq_read_tx_data_flow_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_flow_mode_t * *p_mode*)

The skiq_read_tx_data_flow_mode() function is responsible for returning the current data flow mode for the Tx interface; this can be one of the following:

- skiq_tx_immediate_data_flow_mode, where timestamps are ignored, and data is transmitted as soon as possible.
- skiq_tx_with_timestamps_data_flow_mode, where the FPGA will ensure that the data is sent at the appropriate timestamp.
- skiq_tx_with_timestamps_allow_late_data_flow_mode, where the FPGA will ensure that the data is sent at the appropriate timestamp, but will also send data with timestamps that have already passed.

Note

With skiq_tx_with_timestamps_data_flow_mode, if data arrives when the FPGA's timestamp is greater than the data's associated timestamp, the data is considered late and not transmitted. This is not the case with skiq_tx_with_timestamps_allow_late_data_flow_mode, which will allow late data to be transmitted.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the Tx interface of interest
out	*p_mode*	a pointer to where the current data flow mode will be written

Returns

int32_t status where 0=success, anything else is an error

5.7.5.13 EPIQ_API int32_t skiq_write_tx_data_flow_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_flow_mode_t *mode*)

The skiq_write_tx_data_flow_mode() function is responsible for updating the current data flow mode for the interface; this can be one of the following:

- skiq_tx_immediate_data_flow_mode, where timestamps are ignored, and data is transmitted as soon as possible.
- skiq_tx_with_timestamps_data_flow_mode, where the FPGA will ensure that the data is sent at the appropriate timestamp.
- skiq_tx_with_timestamps_allow_late_data_flow_mode, where the FPGA will ensure that the data is sent at the appropriate timestamp, but will also send data with timestamps that have already passed.

Note

The data flow modes can be changed at any time, but updates are only honored whenever an interface is started through the skiq_start_tx_interface() call.
With skiq_tx_with_timestamps_data_flow_mode, if data arrives when the FPGA's timestamp is greater than the data's associated timestamp, the data is considered late and not transmitted. This is not the case with skiq_tx_with_timestamps_allow_late_data_flow_mode, which will allow late data to be transmitted.

Attention

skiq_tx_with_timestamps_allow_late_data_flow_mode is only available on certain bitstreams; if this mode is set and the card's bitstream doesn't support it, -ENOTSUP is returned.
The late timestamp counter is not updated when in skiq_tx_with_timestamps_allow_late_data_flow_mode, even if the data is transmitted later than its timestamp.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested Tx interface
in	*mode*	the requested data flow mode

Returns

int32_t status where 0=success, anything else is an error

Return values

-ENOTSUP	if skiq_tx_with_timestamps_allow_late_data_flow_mode TX data flow mode is selected and the currently loaded bitfile on the selected card does not support that feature.
-EPERM	if skiq_tx_with_timestamps_allow_late_data_flow_mode TX data flow mode is not selected and the current config for the timestamp base is set to use system timestamps

5.7.5.14 EPIQ_API int32_t skiq_read_tx_transfer_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_transfer_mode_t ∗ *p_transfer_mode*)

The skiq_read_tx_transfer_mode() function is responsible for returning the current transfer mode (skiq_tx_transfer_mode_t) for the Tx interface. This can be either tx synchronous or asynchronous. With skiq_tx_transfer_mode_sync, the skiq_transmit() call blocks until the packet has been received by the FPGA. With skiq_tx_transfer_mode_async, the skiq_transmit() will accept the packet immediately as long as there is adequate space within the buffer to store the block. With skiq_tx_transfer_mode_async, a callback function (see skiq_register_tx_complete_callback() for details) can be registered to notify the application when the transfer to the FPGA has been completed.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the Tx interface of interest
out	*p_transfer_mode*	a pointer to where the current transfer mode will be written

Returns

int32_t status where 0=success, anything else is an error

5.7.5.15 EPIQ_API int32_t skiq_write_tx_transfer_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_transfer_mode_t *transfer_mode*)

The skiq_write_tx_transfer_mode() function is responsible for updating the current transfer mode (skiq_tx_transfer_mode_t) for the Tx interface. Note that this can only be changed if the transmit interface is not currently streaming. If a mode change is attempted while streaming, an error will be returned. With skiq_tx_transfer_mode_sync, the skiq_transmit() call blocks until the packet has been received by the FPGA. With skiq_tx_transfer_mode_async, a call to skiq_transmit() will accept the packet immediately as long as there is adequate space within the buffer to store the block. With skiq_tx_transfer_mode_async, a callback function (see skiq_register_tx_complete_callback() for details) can be registered to notify the application when the transfer to the FPGA has been completed.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested Tx interface
in	*transfer_mode*	the requested transfer flow mode

Returns

int32_t status where 0=success, anything else is an error

5.7.5.16 EPIQ_API int32_t skiq_register_tx_complete_callback (uint8_t *card*, skiq_tx_callback_t *tx_complete*)

The skiq_register_tx_complete_callback() function registers a callback function that should be called when the transfer of a packet at the address provided has been completed. Once the callback function is called the memory location specified by p_data has completed processing.

Note

This callback function is used only when the transmit transfer mode is skiq_tx_transfer_mode_async.

Since

Function signature modified since **v4.0.0** to add private data pointer in callback, see skiq_tx_callback_t for more details.

See Also

skiq_register_tx_enabled_callback

Parameters

in	*card*	card index of the Sidekiq of interest
in	*tx_complete*	pointer to function to call when a packet has finished transfer

Returns

int32_t status where 0=success, anything else is an error

5.7.5.17 EPIQ_API int32_t skiq_register_tx_enabled_callback (uint8_t *card*, skiq_tx_ena_callback_t *tx_ena_cb*)

The skiq_register_tx_enabled_callback() function registers a callback function that should be called when the transmit FIFO is enabled and available to queue packets.

Since

Function added in API **v4.3.0**

See Also

skiq_register_tx_complete_callback

Parameters

in	*card*	card index of the Sidekiq of interest
in	*tx_ena_cb*	[skiq_tx_ena_callback_t] pointer to function to call when FIFO is enabled

Returns

int32_t status where 0=success, anything else is an error

5.7.5.18 EPIQ_API int32_t skiq_write_tx_filter_path (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_filt_t *path*)

The skiq_write_tx_filter_path() function is responsible for selecting from any skiq_filt_t value appropriate for the Sidekiq hardware on the specified Tx interface.

Note

Not all filter options are available for hardware variants. Users may use skiq_read_tx_filters_avail() to determine RF filter path available for a given Sidekiq card.

See Also

skiq_read_tx_filters_avail

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested tx interface
in	*path*	an enum indicating which path is being requested

Returns

int32_t status where 0=success, anything else is an error

5.7.5.19 EPIQ_API int32_t skiq_read_tx_filter_path (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_filt_t * *p_path*)

The skiq_read_tx_filter_path() function is responsible for returning the currently selected RF path on the specified Tx interface.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	the handle of the requested tx interface
out	*p_path*	a pointer to where the current value of the filter path should be written

Returns

int32_t status where 0=success, anything else is an error

5.7.5.20 EPIQ_API int32_t skiq_write_tx_sample_rate_and_bandwidth (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t *rate*, uint32_t *bandwidth*)

The skiq_write_tx_sample_rate_and_bandwidth() function writes the current setting for the rate of transmit samples being transferred from the FPGA to the RFIC. Additionally, the channel bandwidth is also configured.

Note

Rx/Tx sample rates are derived from the same clock so modifications to the Tx sample rate will also update the Rx sample rate to the same value.

See Also

skiq_write_rx_sample_rate_and_bandwidth
skiq_read_tx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
in	*rate*	the new value of the sample rate (in Hertz)

in	*bandwidth*	specifies the channel bandwidth in Hertz

Returns

int32_t status where 0=success, anything else is an error

5.7.5.21 EPIQ_API int32_t skiq_read_tx_sample_rate_and_bandwidth (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t ∗ *p_rate*, double ∗ *p_actual_rate*, uint32_t ∗ *p_bandwidth*, uint32_t ∗ *p_actual_bandwidth*)

The skiq_read_tx_sample_rate_and_bandwidth() function reads the current setting for the rate of transmit samples being transferred from the FPGA to the RFIC and the configured channel bandwidth.

See Also

skiq_write_tx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested rx interface
out	*p_rate*	a pointer to the variable that should be updated with the current sample rate setting (in Hertz) currently set for the specified interface
out	*p_actual_rate*	a pointer to the variable that should be updated with the actual rate of received samples being transferred into the FPGA
out	*p_bandwidth*	a pointer to the variable that is updated with the current channel bandwidth setting (in Hertz)
out	*p_actual_-bandwidth*	a pointer to the variable that is updated with the actual channel bandwidth configured (in Hertz)

Returns

int32_t status where 0=success, anything else is an error

5.7.5.22 EPIQ_API int32_t skiq_transmit (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_block_t ∗ *p_block*, void ∗ *p_user*)

The skiq_transmit() function is responsible for writing a block of I/Q samples to transmit. When running in synchronous mode, this function will block until the FPGA has queued the samples to send. If running in asynchronous mode, the function will return immediately. If the packet has successfully been buffered for transfer, a 0 will be returned. If there is not enough room left in the buffer, SKIQ_TX_ASYNC_SEND_QUEUE_FULL is returned.

The first SKIQ_TX_HEADER_SIZE_IN_WORDS contain metadata associated with transmit packet. Included in the metadata is the desired timestamp to send the samples. If running in skiq_tx_immediate_data_flow_mode the timestamp is ignored and the data is sent immediately. Following the metadata is the block_size (in words) of sample data. The number of words contained in p_samples should match the previously configured Tx block size plus the header size.

The format of the data provided to the transmit call

```
-31-----------------------------------------------------0-
```

```
        word 0   |                                                        |
                 |                      META0 (misc)                      |
        word 1   |                                                        |
                 -31-----------------------------------------------------0-
        word 2   |                                                        |
                 |                    RF TIMESTAMP                        |
        word 3   |                                                        |
                 -31-----------------------------------------------------0-
                 |         12-bit I0_A1        |        12-bit Q0_A1      |
n |-     word 4   | (sign extended to 16 bits   | (sign extended to 16 bits) |
u |              ----------------------------------------------------------
m |              |         12-bit I1_A1        |        12-bit Q1_A1      |
_ |      word 5   | (sign extended to 16 bits   | (sign extended to 16 bits) |
b |              ----------------------------------------------------------
l |              |            ...              |           ...            |
o |              ----------------------------------------------------------
c |      word     |   12-bit Iblock_size_A1     |   12-bit Qblock_size_A1  |
k |       3 +     | (sign extended to 16 bits   | (sign extended to 16 bits) |
s |-  block_size  ----------------------------------------------------------
```

Since

Function signature modified v4.0.0 to take skiq_tx_block_t instead of int32_t pointer for transmit data and a new void pointer argument for user data to be passed back into the callback function if the transmit transfer mode is skiq_tx_transfer_mode_async.

Note

If the caller does not need user data or the transmit transfer mode is skiq_tx_transfer_mode_sync, the caller should pass NULL as p_user.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_start_tx_streaming
skiq_start_tx_streaming_on_1pps
skiq_stop_tx_streaming
skiq_stop_tx_streaming_on_1pps
skiq_read_tx_transfer_mode
skiq_write_tx_transfer_mode
skiq_read_tx_data_flow_mode
skiq_write_tx_data_flow_mode
skiq_tx_block_t

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the desired interface
in	*p_block*	[skiq_tx_block_t] a pointer to the timestamp + I/Q sample data
in	*p_user*	a pointer to user data that is passed back into the callback function if async

Returns

int32_t status where 0=success, SKIQ_TX_ASYNC_SEND_QUEUE_FULL indicates out of room to buffer if in asynchronous mode, anything else is an error

5.7.5.23 EPIQ_API int32_t skiq_read_tx_LO_freq (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t * *p_freq*, double * *p_tuned_freq*)

The skiq_read_tx_LO_freq() function reads the current setting for the LO frequency of the requested tx interface.

See Also

skiq_write_tx_LO_freq

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_freq*	a pointer to the variable that should be updated with the current frequency (in Hertz)
out	*p_tuned_freq*	a pointer to the variable that should be updated with the actual tuned frequency (in Hertz)

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Invalid TX handle specified
-ENODATA	TX LO frequency has not yet been configured

5.7.5.24 EPIQ_API int32_t skiq_write_tx_LO_freq (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t *freq*)

The skiq_write_tx_LO_freq() function writes the current setting for the LO frequency of the requested tx interface.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

See Also

skiq_read_tx_LO_freq

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
in	*freq*	the new value for the LO freq (in Hertz)

Returns

int32_t status where 0=success, anything else is an error

5.7.5.25 EPIQ_API int32_t skiq_enable_tx_tone (uint8_t *card*, skiq_tx_hdl_t *hdl*)

The skiq_enable_tx_tone() function configures the RFIC to send out a single cycle of a CW tone.

Note

The RFIC is responsible generating the tone. There is no reliance on the FPGA or software for this functionality. However, a user must call skiq_start_tx_streaming() to enable the transmitter.

See Also

skiq_disable_tx_tone
skiq_read_tx_tone_freq
skiq_read_tx_tone_freq_offset
skiq_write_tx_tone_freq_offset

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface

Returns

int32_t status where 0=success, anything else is an error

5.7.5.26 EPIQ_API int32_t skiq_disable_tx_tone (uint8_t *card*, skiq_tx_hdl_t *hdl*)

The skiq_disable_tx_tone() function disables the CW tone from being sent out when the transmitter is enabled.

Note

A user must also call skiq_stop_tx_streaming() to disable the transmitter.

See Also

skiq_enable_tx_tone
skiq_read_tx_tone_freq
skiq_read_tx_tone_freq_offset
skiq_write_tx_tone_freq_offset

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface

Returns

int32_t status where 0=success, anything else is an error

5.7.5.27 EPIQ_API int32_t skiq_read_tx_tone_freq (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t * *p_freq*)

The skiq_read_tx_tone_freq() function returns the LO frequency of the TX test tone.

Since

Function added in API **v4.2.0**

See Also

skiq_enable_tx_tone
skiq_disable_tx_tone
skiq_read_tx_tone_freq_offset
skiq_write_tx_tone_freq_offset

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_freq*	pointer to where to store the frequency (in Hz) of the test tone

Returns

int32_t status where 0=success, anything else is an error

5.7.5.28 EPIQ_API int32_t skiq_read_tx_tone_freq_offset (uint8_t *card*, skiq_tx_hdl_t *hdl*, int32_t * *p_freq_offset*)

The skiq_read_tx_tone_freq_offset() function returns the the TX test tone offset relative to the configured TX LO frequency.

Since

Function added in API **v4.9.0**

See Also

skiq_enable_tx_tone
skiq_disable_tx_tone
skiq_read_tx_tone_freq
skiq_write_tx_tone_freq_offset

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_freq_offset*	pointer to where to store the frequency (in Hz) offset

Returns

int32_t status where 0=success, anything else is an error

Return values

-ERANGE	specified card index is out of range
-ENODEV	specified card has not been initialized
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.7.5.29 EPIQ_API int32_t skiq_write_tx_tone_freq_offset (uint8_t *card*, skiq_tx_hdl_t *hdl*, int32_t *test_freq_offset*)

The skiq_write_tx_tone_freq_offset() function configures the frequency of the TX test tone offset from the configured TX LO frequency.

Since

Function added in API **v4.9.0**

Note

This is not available for all products
The frequency offset generally needs to fall within the +/- 0.5*sample_rate

See Also

skiq_enable_tx_tone
skiq_disable_tx_tone
skiq_read_tx_tone_freq
skiq_read_tx_tone_freq_offset

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
in	*test_freq_offset*	test tone frequency (in Hz) offset

Returns

int32_t status where 0=success, anything else is an error

Return values

-ERANGE	specified card index is out of range
-ENODEV	specified card has not been initialized
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.7.5.30 EPIQ_API int32_t skiq_write_tx_attenuation (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t *attenuation*)

The skiq_write_tx_attenuation() function configures the attenuation of the transmitter for the Tx handle specified. The value of the attenuation is 0.25 dB steps such that an attenuation value of 4 would equate to 1 dB of actual attenuation. A value of 0 would provide result in 0 attenuation, or maximum transmit power. Valid attenuation settings are queried using skiq_read_parameters().

Note

If the specified attenuation is outside the radio's valid range, the attentuation level is set to the nearest allowed value, the maximum or minimum value.

See Also

skiq_read_tx_attenuation
skiq_read_parameters

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
in	*attenuation*	value of attenuation

Returns

int32_t status where 0=success, anything else is an error

5.7.5.31 EPIQ_API int32_t skiq_read_tx_attenuation (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t * *p_attenuation*)

The skiq_read_tx_attenuation() function reads the attenuation setting of the transmitter for the Tx handle specified. The value of the attenuation is 0.25 dB steps such that an attenuation value of 4 would equate to 1 dB of actual attenuation.

See Also

skiq_read_parameters
skiq_write_tx_attenuation

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_attenuation*	pointer to where to store the attenuation read

Returns

int32_t status where 0=success, anything else is an error

5.7.5.32 EPIQ_API int32_t skiq_read_tx_sample_rate (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t * *p_rate*, double * *p_actual_rate*)

The skiq_read_tx_sample_rate() function reads the current setting for the rate at which samples will be delivered from the FPGA to the RF front end for transmission.

See Also

skiq_read_tx_sample_rate_and_bandwidth
skiq_write_tx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_rate*	a pointer to the variable that should be updated with the actual sample rate (in Hertz) currently set for the D/A converter
out	*p_actual_rate*	a pointer to the variable that should be updated with the actual sample rate (in Hertz) currently set

Returns

int32_t status where 0=success, anything else is an error

5.7.5.33 EPIQ_API int32_t skiq_read_tx_block_size (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t * *p_block_size_in_words*)

The skiq_read_tx_block_size() function reads the current setting for the block size of transmit packets.

Note

The block size is represented in words and does not include the header size, it accounts only for the number of samples. The total Tx packet size includes both the header size and block size.

See Also

skiq_write_tx_block_size

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_block_size_in-_words*	a pointer to the variable that should be updated with current Tx block size

Returns

int32_t status where 0=success, anything else is an error

5.7.5.34 EPIQ_API int32_t skiq_write_tx_block_size (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t *block_size_in_words*)

The skiq_write_tx_block_size() function configures the block size of transmit packets.

Note

The block size is represented in words and is the size (in words) of the IQ samples for each channel, not including the metadata. When using packed mode, this is the number of words (not number of samples) in the payload, not including the metadata. Also, while in packed mode, the value specified must result in an even number of samples included in a block. For instance, a block size of 252 ∗ 4/3 = 336 samples per block of data, which is a valid configuration. A block size of 508 ∗ 4/3 - 677.3 samples per block would be invalid.

Attention

The validity of the configuration will not be confirmed until start streaming is called.

Note

This must be set prior to the Tx interface being started. If set after the Tx interface has been started, the setting will be stored but will not be used until the interface is stopped and re-started.

See Also

skiq_read_tx_block_size

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
in	*block_size_in_-words*	number of words to configure the Tx block size

Returns

int32_t status where 0=success, anything else is an error

5.7.5.35 EPIQ_API int32_t skiq_read_tx_num_underruns (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t ∗ *p_num_underrun*)

The skiq_read_tx_num_underruns() function reads the current number of Tx underruns observed by the FPGA. This value is reset only when calling skiq_start_tx_streaming().

Warning

This number is only valid if running with Tx data flow mode set to skiq_tx_immediate_data_flow_mode.

See Also

skiq_read_tx_data_flow_mode
skiq_write_tx_data_flow_mode
skiq_read_tx_num_late_timestamps

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_num_underrun*	a pointer to the variable that is updated with the number of underruns observed since starting streaming

Returns

int32_t status where 0=success, anything else is an error

5.7.5.36 EPIQ_API int32_t skiq_read_tx_num_late_timestamps (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t ∗ *p_num_late*)

The skiq_read_tx_num_late_timestamps() function reads the current number of "late" Tx timestamps observed by the FPGA. When the FPGA encounters a Tx timestamp that has occurred in the past, the FPGA Tx FIFO is flushed of all packets and a counter is incremented. This function returns the count of how many times the FIFO was flushed due to a timestamp in the past. The value is reset only after calling skiq_stop_tx_streaming().

Warning

The late timestamp count value is only valid if running with Tx data flow mode set to skiq_tx_with_timestamps_data_flow_mode and not skiq_tx_immediate_data_flow_mode or skiq_tx_with_timestamps_allow_late_data_flow_mode.

Attention

The late timestamp counter is not updated when in skiq_tx_with_timestamps_allow_late_data_flow_mode, even if the data is transmitted later than its timestamp.

See Also

skiq_read_tx_data_flow_mode
skiq_write_tx_data_flow_mode
skiq_read_tx_num_underruns

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_num_late*	a pointer to the variable that is updated with the number of times the FIFO is flushed due to a "late" timestamp

Returns

int32_t status where 0=success, anything else is an error

5.7.5.37 EPIQ_API int32_t skiq_write_tx_fir_gain (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_fir_gain_t *gain*)

The skiq_write_tx_fir_gain() function is responsible for configuring the gain of the Tx FIR filter. The Tx FIR filter is used in configuring the Tx channel bandwidth.

See Also

skiq_read_tx_fir_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the Tx interface to access
in	*gain*	[skiq_tx_fir_gain_t] gain of the filter

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.7.5.38 EPIQ_API int32_t skiq_read_tx_fir_gain (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_fir_gain_t * *p_gain*)

The skiq_read_tx_fir_gain() function is responsible for reading the gain of the Tx FIR filter. The Tx FIR filter is used in configuring the Tx channel bandwidth.

See Also

skiq_write_tx_fir_gain

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the Tx interface to access
out	*p_gain*	[skiq_tx_fir_gain_t] pointer to where to store the gain setting

Returns

int32_t status of the operation (0=success, anything else is an error code)

5.7.5.39 EPIQ_API int32_t skiq_read_num_tx_threads (uint8_t *card*, uint8_t * *p_num_threads*)

The skiq_read_num_tx_threads() function is responsible for returning the number of threads used to transfer data when operating in asynchronous mode.

See Also

skiq_write_num_tx_threads
skiq_read_tx_thread_priority
skiq_write_tx_thread_priority

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_num_threads*	pointer to where to store the number of threads

Returns

int32_t status where 0=success, anything else is an error

5.7.5.40 EPIQ_API int32_t skiq_write_num_tx_threads (uint8_t *card*, uint8_t *num_threads*)

The skiq_write_num_tx_threads() function is responsible for updating the number of threads used to transfer data when operating in asynchronous mode. This must be set prior to the Tx interface being started. If set after the Tx interface has been started, the setting will be stored but will not be used until the interface is stopped and re-started.

See Also

skiq_read_num_tx_threads
skiq_read_tx_thread_priority
skiq_write_tx_thread_priority

Parameters

in	*card*	card index of the Sidekiq of interest
in	*num_threads*	number of threads to use when running in Tx asynchronous mode

Returns

int32_t status where 0=success, anything else is an error

5.7.5.41 EPIQ_API int32_t skiq_read_tx_thread_priority (uint8_t *card*, int32_t * *p_priority*)

The skiq_read_tx_thread_priority() function is responsible for returning the priority of the threads when operating in asynchronous mode.

See Also

skiq_read_num_tx_threads
skiq_write_num_tx_threads
skiq_write_tx_thread_priority

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_priority*	pointer to where to store the priority of the TX threads

Returns

int32_t status where 0=success, anything else is an error

5.7.5.42 EPIQ_API int32_t skiq_write_tx_thread_priority (uint8_t *card*, int32_t *priority*)

The skiq_write_tx_thread_priority() function is responsible for updating the priority of the threads used to transfer data when operating in asynchronous mode. This must be set prior to the Tx interface being started. If set after the Tx interface has been started, the setting will be stored but will not be used until the interface is stopped and re-started.

See Also

skiq_read_num_tx_threads
skiq_write_num_tx_threads
skiq_read_tx_thread_priority

Parameters

in	*card*	card index of the Sidekiq of interest
in	*priority*	TX thread priority

Returns

int32_t status where 0=success, anything else is an error

5.7.5.43 EPIQ_API int32_t skiq_read_num_tx_chans (uint8_t *card*, uint8_t ∗ *p_num_tx_chans*)

The skiq_read_num_tx_chans() function is responsible for returning the number of Tx channels supported for the Sidekiq card of interest. The handle for the first Tx interface is skiq_tx_hdl_A1 and increments from there.

See Also

skiq_tx_hdl_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_num_tx_-chans*	pointer to the number of Tx channels

Returns

int32_t status where 0=success, anything else is an error

5.7.5.44 EPIQ_API int32_t skiq_read_tx_iq_resolution (uint8_t *card,* uint8_t ∗ *p_dac_res*)

The skiq_read_tx_iq_resolution() function is responsible for returning the resolution (in bits) per TX (DAC) IQ sample.

Since

Function added in API **v4.2.0**

See Also

skiq_transmit
skiq_tx_block_t

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_dac_res*	pointer to the number of DAC bits

Returns

int32_t status where 0=success, anything else is an error

5.7.5.45 EPIQ_API int32_t skiq_read_tx_LO_freq_range (uint8_t *card,* uint64_t ∗ *p_max,* uint64_t ∗ *p_min*)

The skiq_read_tx_LO_freq_range() function allows an application to obtain the maximum and minimum frequencies that a Sidekiq can tune to transmit. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_max_tx_LO_freq
skiq_read_min_tx_LO_freq
skiq_read_parameters
skiq_tx_param_t::lo_freq_min
skiq_tx_param_t::lo_freq_max

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_max*	pointer to update with maximum LO frequency
out	*p_min*	pointer to update with minimum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.7.5.46 EPIQ_API int32_t skiq_read_max_tx_LO_freq (uint8_t *card,* uint64_t ∗ *p_max*)

The skiq_read_max_tx_LO_freq() function allows an application to obtain the maximum frequency that a Sidekiq can tune to transmit. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_tx_LO_freq_range
skiq_read_min_tx_LO_freq
skiq_read_parameters
skiq_tx_param_t::lo_freq_max

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_max*	pointer to update with maximum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.7.5.47 EPIQ_API int32_t skiq_read_min_tx_LO_freq (uint8_t *card*, uint64_t * *p_min*)

The skiq_read_min_tx_LO_freq() function allows an application to obtain minimum frequency that a Sidekiq can tune to transmit at. This information may also be accessed using skiq_read_parameters().

See Also

skiq_read_tx_LO_freq_range
skiq_read_max_tx_LO_freq
skiq_read_parameters
skiq_tx_param_t::lo_freq_min

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_min*	pointer to update with minimum LO frequency

Returns

int32_t status where 0=success, anything else is an error

5.7.5.48 EPIQ_API int32_t skiq_read_tx_quadcal_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_quadcal_mode_t * *p_mode*)

The skiq_read_tx_quadcal_mode() function reads the TX quadrature calibration algorithm mode.

Since

Function added in API **v4.6.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest
out	*p_mode*	[skiq_tx_quadcal_mode_t] the currently set value of the TX quadrature calibration mode setting

Returns

int32_t status where 0=success, anything else is an error

5.7.5.49 EPIQ_API int32_t skiq_write_tx_quadcal_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_tx_quadcal_mode_t *mode*)

The skiq_write_tx_quadcal_mode() function writes the TX quadrature calibration algorithm mode. If automatic mode is configured, writing the TX LO frequency may result in the TX quadrature calibration algorithm to be run, resulting in the transmission of calibration waveforms which can take a significant amount of time to complete. If manual mode is configured, it is the user's responsibility to determine when to run the TX quadrature calibration algorithm via skiq_run_tx_quadcal().

Since

Function added in API **v4.6.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest
in	*mode*	[skiq_tx_quadcal_mode_t] TX quadrature calibration mode to configure

Returns

int32_t status where 0=success, anything else is an error

5.7.5.50 EPIQ_API int32_t skiq_run_tx_quadcal (uint8_t *card*, skiq_tx_hdl_t *hdl*)

The skiq_run_tx_quadcal() performs the TX quadrature calibration algorithm based on the current RFIC settings.

Note

This quadrature calibration may take some time to complete. Additionally, running of the TX quadrature algorithm results in transmissions of calibration waveforms, resulting in the appearance of erroneous transmissions in the spectrum during execution of the algorithm. Streaming RX or TX while running the TX quadrature algorithm will result in a momentary gap in received and/or transmitted samples. It is recommended that this is ran after the desired Tx LO frequency has been configured.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.
In the case of Sidekiq X2, calibration is performed on all TX handles, regardless of the handle specified.

Since

Function added in API **v4.6.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest

Returns

int32_t status where 0=success, anything else is an error

5.7.5.51 EPIQ_API int32_t skiq_read_tx_analog_filter_bandwidth (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t * *p_bandwidth*)

The skiq_read_tx_analog_filter_bandwidth() function reads the current setting for the TX analog filter bandwidth.

Since

Function added in 4.17.0

Note

that this value is automatically updated when the channel bandwidth is changed
This is not available for all products

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
out	*p_bandwidth*	pointer to the variable that should be updated with the actual bandwidth of the analog filter bandwidth

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.7.5.52 EPIQ_API int32_t skiq_write_tx_analog_filter_bandwidth (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint32_t *bandwidth*)

The skiq_write_tx_analog_filter_bandwidth() function writes the current bandwidth of the analog filter.

Since

Function added in 4.17.0

Note

that this value is overwritten when the bandwidth is configured with skiq_write_rx_sample_rate_and_-bandwidth
This is not available for all products
not all bandwidth settings are valid and actual setting can be queried
For AD9361 products, the analog filter bandwidth is typically set to the configured channel bandwidth and is automatically configured to this value when the sample rate and channel bandwidth is configured. This function allows the analog filter bandwidth to be overwritten, where the corner frequency of the 3rd order Butterworth filter is set to 1.6x of half the specified bandwidth.

See Also

skiq_write_tx_sample_rate_and_bandwidth

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] the handle of the requested tx interface
in	*bandwidth*	specifies the analog filter bandwidth in Hertz

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_mode` is NULL
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality

5.8 Fast Frequency Hopping Functions and Definitions

These functions and definitions are related to configuring and exercising the fast frequency hopping capabilities of the Sidekiq SDR.

Enumerations

- enum skiq_freq_tune_mode_t { skiq_freq_tune_mode_standard =0, skiq_freq_tune_mode_hop_immediate, skiq_freq_tune_mode_hop_on_timestamp }

  Frequency Tune mode. Note that not all products support all configurations.

Functions

- EPIQ_API int32_t skiq_write_rx_freq_tune_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_freq_tune_mode_t mode)
- EPIQ_API int32_t skiq_read_rx_freq_tune_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_freq_tune_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_tx_freq_tune_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_freq_tune_mode_t mode)
- EPIQ_API int32_t skiq_read_tx_freq_tune_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_freq_tune_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_rx_freq_hop_list (uint8_t card, skiq_rx_hdl_t hdl, uint16_t num_freq, uint64_t freq_list[ ], uint16_t initial_index)
- EPIQ_API int32_t skiq_read_rx_freq_hop_list (uint8_t card, skiq_rx_hdl_t hdl, uint16_t ∗p_num_freq, uint64_t freq_list[SKIQ_MAX_NUM_FREQ_HOPS])
- EPIQ_API int32_t skiq_write_tx_freq_hop_list (uint8_t card, skiq_tx_hdl_t hdl, uint16_t num_freq, uint64_t freq_list[ ], uint16_t initial_index)
- EPIQ_API int32_t skiq_read_tx_freq_hop_list (uint8_t card, skiq_tx_hdl_t hdl, uint16_t ∗p_num_freq, uint64_t freq_list[SKIQ_MAX_NUM_FREQ_HOPS])
- EPIQ_API int32_t skiq_write_next_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t freq_index)
- EPIQ_API int32_t skiq_write_next_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t freq_index)
- EPIQ_API int32_t skiq_perform_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint64_t rf_timestamp)
- EPIQ_API int32_t skiq_perform_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint64_t rf_timestamp)
- EPIQ_API int32_t skiq_read_curr_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t ∗p_hop_index, uint64_t ∗p_curr_freq)
- EPIQ_API int32_t skiq_read_next_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t ∗p_hop_index, uint64_t ∗p_curr_freq)
- EPIQ_API int32_t skiq_read_curr_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t ∗p_hop_index, uint64_t ∗p_curr_freq)
- EPIQ_API int32_t skiq_read_next_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t ∗p_hop_index, uint64_t ∗p_curr_freq)

5.8.1 Detailed Description

These functions and definitions are related to configuring and exercising the fast frequency hopping capabilities of the Sidekiq SDR.

5.8.2 Enumeration Type Documentation

5.8.2.1 enum skiq_freq_tune_mode_t

Frequency Tune mode. Note that not all products support all configurations.

See Also

skiq_write_rx_freq_tune_mode
skiq_read_rx_freq_tune_mode
skiq_write_tx_freq_tune_mode
skiq_read_tx_freq_tune_mode
skiq_write_rx_freq_hop_list
skiq_read_rx_freq_hop_list
skiq_write_tx_freq_hop_list
skiq_read_tx_freq_hop_list
skiq_perform_rx_freq_hop
skiq_perform_tx_freq_hop
skiq_read_curr_rx_freq_hop
skiq_read_curr_tx_freq_hop

Enumerator

skiq_freq_tune_mode_standard LO frequency adjusted with either skiq_write_rx_LO_freq() or skiq_write_tx_LO_freq() depending on the handle in use.

skiq_freq_tune_mode_hop_immediate hop list index used to control LO, tuning happens ASAP

skiq_freq_tune_mode_hop_on_timestamp hop list index used to control LO, tuning initiated on timestamp

Definition at line 884 of file sidekiq_types.h.

5.8.3 Function Documentation

5.8.3.1 EPIQ_API int32_t skiq_write_rx_freq_tune_mode (uint8_t *card,* skiq_rx_hdl_t *hdl,* skiq_freq_tune_mode_t *mode*)

The skiq_write_rx_freq_tune_mode() function configures the frequency tune mode for the handle specified.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this configures the tune mode for both receive and transmit of the RFIC specified by the RX handle (ex. RX A1/A2/C1 configures RFIC A)
For Sidekiq X2, skiq_freq_tune_mode_hop_on_timestamp is not supported. Additionally, skiq_rx_hdl_B1 is not supported.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*mode*	[skiq_freq_tune_mode_t] tune mode

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENOTSUP	Mode is not supported by hardware
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.2 EPIQ_API int32_t skiq_read_rx_freq_tune_mode (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_freq_tune_mode_t ∗ *p_mode*)

The skiq_read_rx_freq_tune_mode() function reads the configured frequency tune mode for the handle specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_mode*	[skiq_freq_tune_mode_t] pointer to tune mode

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.3 EPIQ_API int32_t skiq_write_tx_freq_tune_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_freq_tune_mode_t *mode*)

The skiq_write_tx_freq_tune_mode() function configures the frequency tune mode for the handle specified.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this configures the tune mode for both receive and transmit of the RFIC specified by the TX handle (ex. TX A1/A2 configures RFIC A)
For Sidekiq X2, skiq_freq_tune_mode_hop_on_timestamp is not supported.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest
in	*mode*	[skiq_freq_tune_mode_t] tune mode

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENOTSUP	Mode is not supported by hardware
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.4 EPIQ_API int32_t skiq_read_tx_freq_tune_mode (uint8_t *card*, skiq_tx_hdl_t *hdl*, skiq_freq_tune_mode_t ∗ *p_mode*)

The skiq_read_tx_freq_tune_mode() function reads the configured frequency tune mode for the handle specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
out	*p_mode*	[skiq_freq_tune_mode_t] pointer to tune mode

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.5 EPIQ_API int32_t skiq_write_rx_freq_hop_list (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t *num_freq*, uint64_t *freq_list[ ]*, uint16_t *initial_index*)

The skiq_write_rx_freq_hop_list() function configures the frequency hop list to the values specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*num_freq*	number of frequencies included in freq_list; this value cannot exceed SKIQ_MAX_NUM_FREQ_HOPS
in	*freq_list*	list of frequencies supported in hopping list
in	*initial_index*	initial index of frequency for first hop

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range or # freqs out of range or initial index out of range
-ERANGE	Number of frequencies is not less than SKIQ_MAX_NUM_FREQ_HOPS
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EINVAL	freq_list contains invalid frequency
non-zero	Unspecified error occurred

5.8.3.6 EPIQ_API int32_t skiq_read_rx_freq_hop_list (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t ∗ *p_num_freq*, uint64_t *freq_list[SKIQ_MAX_NUM_FREQ_HOPS]*)

The skiq_read_rx_freq_hop_list() function populates the frequency hop list with the frequency values previously specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_num_freq*	pointer to number of frequencies included in list
out	*freq_list*	hopping list currently configured; this list should be able to hold at least SKIQ_MAX_NUM_FREQ_HOPS

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.7 EPIQ_API int32_t skiq_write_tx_freq_hop_list (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t *num_freq*, uint64_t *freq_list[ ]*, uint16_t *initial_index*)

The skiq_write_tx_freq_hop_list() function configures the frequency hop list to the values specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
in	*num_freq*	number of frequencies included in freq_list this value cannot exceed SKIQ_MAX_NUM_FREQ_HOPS
in	*freq_list*	list of frequencies supported in hopping list
in	*initial_index*	initial index of frequency for first hop

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range or # freqs out of range or initial index out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

-EINVAL	freq_list contains invalid frequency
non-zero	Unspecified error occurred

5.8.3.8 EPIQ_API int32_t skiq_read_tx_freq_hop_list (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t * *p_num_freq*, uint64_t *freq_list[SKIQ_MAX_NUM_FREQ_HOPS]*)

The skiq_read_tx_freq_hop_list() function populates the frequency hop list with the values previously specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
out	*p_num_freq*	pointer to number of frequencies included in list
out	*freq_list*	hopping list currently configured; this list should be able to hold at least SKIQ_MAX_NUM_FREQ_HOPS

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.9 EPIQ_API int32_t skiq_write_next_rx_freq_hop (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t *freq_index*)

The skiq_write_next_rx_freq_hop() function performs the various configuration required to support the next frequency hop but does not execute the hop until skiq_perform_rx_freq_hop() is called.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this updates both the RX and TX LO frequency.
For any radio based on the AD9361 RF IC (mPCIe, m.2, Z2), when operating in the skiq_freq_tune_mode_hop_on_timestamp, this updates both the RX and TX LO frequency based on the index specified.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*freq_index*	index into hopping list of frequency to configure

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range or freq index out of range
-ENODEV	Requested card index is not initialized
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
non-zero	Unspecified error occurred

5.8.3.10 EPIQ_API int32_t skiq_write_next_tx_freq_hop (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t *freq_index*)

The skiq_write_next_tx_freq_hop() function performs the various configuration required to support the next frequency hop but does not execute the hop until skiq_perform_tx_freq_hop() is called.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this updates both the RX and TX LO frequency.
For any radio based on the AD9361 RF IC (mPCIe, m.2, Z2), when operating in the skiq_freq_tune_-mode_hop_on_timestamp, this updates both the RX and TX LO frequency based on the index specified.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest
in	*freq_index*	index into hopping list of frequency to configure

Returns

int32_t

Return values

0	successful

-ERANGE	Requested card index is out of range or freq index out of range
-ENODEV	Requested card index is not initialized
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
non-zero	Unspecified error occurred

5.8.3.11 EPIQ_API int32_t skiq_perform_rx_freq_hop (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *rf_timestamp*)

The skiq_perform_rx_freq_hop() function performs the frequency hop for the handle specified.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this updates both the RX and TX LO frequency.
For any radio based on the AD9361 RF IC (mPCIe, m.2, Z2), when operating in the skiq_freq_tune_mode_hop_on_timestamp, this updates both the RX and TX LO frequency based on the index specified. if operating in skiq_freq_tune_mode_hop_on_timestamp and a rf_timestamp that has already passed is specified, the frequency hop will be executed immediately. If running in skiq_freq_tune_mode_hop_immediate, the timestamp parameter is ignored.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*rf_timestamp*	timestamp to execute the hop (only for skiq_freq_tune_mode_hop_on_timestamp)

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.12 EPIQ_API int32_t skiq_perform_tx_freq_hop (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t *rf_timestamp*)

The skiq_perform_tx_freq_hop() function performs the frequency hop for the handle specified.

Since

Function added in API **v4.10.0**

Note

For Sidekiq X4, this updates both the RX and TX LO frequency.
For any radio based on the AD9361 RF IC (mPCIe, m.2, Z2), when operating in the skiq_freq_tune_mode_hop_on_timestamp, this updates both the RX and TX LO frequency based on the index specified. if operating in skiq_freq_tune_mode_hop_on_timestamp and a rf_timestamp that has already passed is specified, the frequency hop will be executed immediately. If running in skiq_freq_tune_mode_hop_immediate, the timestamp parameter is ignored.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
in	*rf_timestamp*	timestamp to execute the hop (only for skiq_freq_tune_mode_hop_on_timestamp)

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized or an error occurred while applying hopping config to RF IC
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.13 EPIQ_API int32_t skiq_read_curr_rx_freq_hop (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint16_t ∗ *p_hop_index*, uint64_t ∗ *p_curr_freq*)

The skiq_read_curr_rx_freq_hop() function reads the current frequency hopping configuration for the handle specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_hop_index*	pointer to the current hopping index
out	*p_curr_freq*	pointer to the current frequency

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EINVAL	Invalid pointers provided
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.14 EPIQ_API int32_t skiq_read_next_rx_freq_hop (uint8_t *card,* skiq_rx_hdl_t *hdl,* uint16_t ∗ *p_hop_index,* uint64_t ∗ *p_curr_freq*)

The skiq_read_next_rx_freq_hop() function reads the next frequency hopping configuration for the handle specified. This is the configuration that will be applied the next "perform hop" function is called.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_hop_index*	pointer to the current hopping index
out	*p_curr_freq*	pointer to the current frequency

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EINVAL	Invalid pointers provided
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.15 EPIQ_API int32_t skiq_read_curr_tx_freq_hop (uint8_t *card,* skiq_tx_hdl_t *hdl,* uint16_t ∗ *p_hop_index,* uint64_t ∗ *p_curr_freq*)

The skiq_read_curr_tx_freq_hop() function reads the current frequency hopping configuration for the handle specified.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
out	*p_hop_index*	pointer to the current hopping index
out	*p_curr_freq*	pointer to the current frequency

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EINVAL	Invalid pointers provided
-EPROTO	Tune mode is not hopping
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.8.3.16 EPIQ_API int32_t skiq_read_next_tx_freq_hop (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint16_t * *p_hop_index*, uint64_t * *p_curr_freq*)

The skiq_read_next_tx_freq_hop() function reads the next frequency hopping configuration for the handle specified. This is the configuration that will be applied the next "perform hop" function is called.

Since

Function added in API **v4.10.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] receive handle of interest
out	*p_hop_index*	pointer to the current hopping index
out	*p_curr_freq*	pointer to the current frequency

Returns

int32_t

Return values

0	successful
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EINVAL	Invalid pointers provided
-EPROTO	Tune mode is not hopping

-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.9 FPGA Functions and Definitions

These functions and definitions are related to communicating and exercising the FPGA capabilities of the Sidekiq SDR.

Macros

- #define SKIQ_START_USER_FPGA_REG_ADDR 0x00008700

  SKIQ_START_USER_FPGA_REG_ADDR is first address available in the FPGA memory map that can be user defined. These 32-bit register addresses increment by 4 bytes
- #define SKIQ_END_USER_FPGA_REG_ADDR 0x00008FFF

  SKIQ_END_USER_FPGA_REG_ADDR is last address of the last FPGA register available in the FPGA memory map that can be user defined.

Enumerations

- enum skiq_fpga_tx_fifo_size_t {
  skiq_fpga_tx_fifo_size_unknown = 0, skiq_fpga_tx_fifo_size_4k = 1, skiq_fpga_tx_fifo_size_8k = 2, skiq_fpga_tx_fifo_size_16k = 3,
  skiq_fpga_tx_fifo_size_32k = 4, skiq_fpga_tx_fifo_size_64k = 5 }

  FPGA Tx FIFO Size. The FIFO size is the number of packets the FPGA can hold prior to actually transmitting the data.

Functions

- EPIQ_API int32_t skiq_write_iq_pack_mode (uint8_t card, bool mode)
- EPIQ_API int32_t skiq_read_iq_pack_mode (uint8_t card, bool *p_mode)
- EPIQ_API int32_t skiq_write_iq_order_mode (uint8_t card, skiq_iq_order_t mode)
- EPIQ_API int32_t skiq_read_iq_order_mode (uint8_t card, skiq_iq_order_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_data_src (uint8_t card, skiq_rx_hdl_t hdl, skiq_data_src_t src)
- EPIQ_API int32_t skiq_read_rx_data_src (uint8_t card, skiq_rx_hdl_t hdl, skiq_data_src_t *p_src)
- EPIQ_API int32_t skiq_read_fpga_semantic_version (uint8_t card, uint8_t *p_major, uint8_t *p_minor, uint8_t *p_patch)
- EPIQ_API int32_t skiq_read_fpga_tx_fifo_size (uint8_t card, skiq_fpga_tx_fifo_size_t *p_tx_fifo_size)
- EPIQ_API int32_t skiq_write_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t data)
- EPIQ_API int32_t skiq_read_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t *p_data)
- EPIQ_API int32_t skiq_write_and_verify_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t data)
- EPIQ_API int32_t skiq_prog_fpga_from_file (uint8_t card, FILE *fp)
- EPIQ_API int32_t skiq_prog_fpga_from_flash (uint8_t card)
- EPIQ_API int32_t skiq_save_fpga_config_to_flash (uint8_t card, FILE *p_file)
- EPIQ_API int32_t skiq_verify_fpga_config_from_flash (uint8_t card, FILE *p_file)
- EPIQ_API int32_t skiq_read_golden_fpga_present_in_flash (uint8_t card, uint8_t *p_present)
- EPIQ_API int32_t skiq_read_last_tx_timestamp (uint8_t card, skiq_tx_hdl_t hdl, uint64_t *p_last_timestamp)
- EPIQ_API int32_t skiq_prog_fpga_from_flash_slot (uint8_t card, uint8_t slot)

  This function is responsible for programming the FPGA from an image stored in flash at the specified slot.

5.9.1 Detailed Description

These functions and definitions are related to communicating and exercising the FPGA capabilities of the Sidekiq SDR.

5.9.2 Macro Definition Documentation

5.9.2.1 #define SKIQ_START_USER_FPGA_REG_ADDR 0x00008700

SKIQ_START_USER_FPGA_REG_ADDR is first address available in the FPGA memory map that can be user defined. These 32-bit register addresses increment by 4 bytes

Definition at line 450 of file sidekiq_api.h.

5.9.2.2 #define SKIQ_END_USER_FPGA_REG_ADDR 0x00008FFF

SKIQ_END_USER_FPGA_REG_ADDR is last address of the last FPGA register available in the FPGA memory map that can be user defined.

Definition at line 454 of file sidekiq_api.h.

5.9.3 Enumeration Type Documentation

5.9.3.1 enum skiq_fpga_tx_fifo_size_t

FPGA Tx FIFO Size. The FIFO size is the number of packets the FPGA can hold prior to actually transmitting the data.

See Also

skiq_read_fpga_tx_fifo_size

Enumerator

skiq_fpga_tx_fifo_size_unknown FPGA versions prior to 2.0 did not support reporting FIFO size
skiq_fpga_tx_fifo_size_4k 4k 32-bit words deep
skiq_fpga_tx_fifo_size_8k 8k 32-bit words deep
skiq_fpga_tx_fifo_size_16k 16k 32-bit words deep
skiq_fpga_tx_fifo_size_32k 32k 32-bit words deep
skiq_fpga_tx_fifo_size_64k 64k 32-bit words deep

Definition at line 720 of file sidekiq_types.h.

5.9.4 Function Documentation

5.9.4.1 EPIQ_API int32_t skiq_write_iq_pack_mode (uint8_t *card,* bool *mode*)

The skiq_write_iq_pack_mode() function is responsible for setting whether or not the IQ samples being received/transmitted and to/from the FPGA to/from the CPU should be packed/compressed before being sent. This allows four 12-bit complex I/Q samples to be transferred in three 32-bit words, increasing the throughput efficiency of the channel. An interface defaults to using un-packed mode if the skiq_write_iq_pack_mode() is not called.

Note

That this can be changed at any time, but updates are only honored whenever streaming is started.

If the pack "mode" is set to false, the behavior is to have the I/Q sent up as two's complement, sign-extended, little-endian, unpacked in the following format:

```
       -31-------------------------------------------------------0-
        |         12-bit I0           |        12-bit Q0          |
word 0  | (sign extended to 16 bits   | (sign extended to 16 bits) |
        ------------------------------------------------------------
        |         12-bit I1           |        12-bit Q1          |
word 1  | (sign extended to 16 bits   | (sign extended to 16 bits) |
        ------------------------------------------------------------
        |         12-bit I2           |        12-bit Q2          |
word 2  |  (sign extended to 16 bits  | (sign extended to 16 bits) |
        ------------------------------------------------------------
        |           ...               |          ...              |
        ------------------------------------------------------------
```

When the mode is set to true, then the 12-bit samples are packed in to make optimal use of the available bits, and packed as follows:

```
       -31-------------------------------------------------------0-
word 0  |I0b11|...|I0b0|Q0b11|................|Q0b0|I1b11|...|I1b4|
        ------------------------------------------------------------
word 1  |I1b3|...|I1b0|Q1b11|...|Q1b0|I2b11|...|I2b0|Q2b11|...|Q2b8|
       -31-------------------------------------------------------0-
word 2  |Q2b7|...|Q2b0|I3b11|................|I3b0|Q1311|....|Q3b4|
        ------------------------------------------------------------
        |           ...               |          ...              |
        ------------------------------------------------------------
```

(with the above sequence repeated every three words)

Once the packed I/Q samples are received up in the CPU there are extra cycles needed to de-compress/un-pack them. However, for cases where an application simply needs to transfer a large block of contiguous I/Q samples up to the CPU for non-real time post processing, this will increase the bandwidth without sacrificing dynamic range.

Warning

I/Q pack mode conflicts with skiq_rx_stream_mode_low_latency. As such, caller may not configure a card to use both packed I/Q mode and RX low latency mode at the same time. This function will return an error (-EPERM) if caller sets mode to true and skiq_rx_stream_mode_low_latency is currently selected.

See Also

skiq_read_iq_pack_mode
skiq_read_rx_stream_mode
skiq_write_rx_stream_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*mode*	false=use normal (non-packed) I/Q mode (default) true=use packed I/Q mode

Returns

int32_t status where 0=success, anything else is an error

5.9.4.2 EPIQ_API int32_t skiq_read_iq_pack_mode (uint8_t *card*, bool ∗ *p_mode*)

The skiq_read_iq_pack_mode() function is responsible for retrieving the current pack mode setting for the Sidekiq card.

See Also

skiq_write_iq_pack_mode

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_mode*	the currently set value of the pack mode setting

Returns

int32_t status where 0=success, anything else is an error

5.9.4.3 EPIQ_API int32_t skiq_write_iq_order_mode (uint8_t *card*, skiq_iq_order_t *mode*)

The skiq_write_iq_order_mode() function is responsible for setting the ordering of the complex samples for the Sidekiq card. Each sample is little-endian, twos-complement, signed, and sign-extended from 12 to 16-bits. (when appropriate for the product) By default samples are received/transmitted as I/Q pairs with 'Q' sample occurring first, followed by the 'I' sample, as depicted.

```
             skiq_iq_order_qi: (default)              skiq_iq_order_iq:
           -15--------------------------0-      -15--------------------------0-
           |          12-bit Q0_A1        |      |          12-bit I0_A1        |
index 0    | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
           -------------------------------      -------------------------------
           |          12-bit I0_A1        |      |          12-bit Q0_A1        |
index 1    | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
           -------------------------------      -------------------------------
           |          12-bit Q1_A1        |      |          12-bit I1_A1        |
index 2    | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
           -------------------------------      -------------------------------
           |          12-bit I1_A1        |      |          12-bit Q1_A1        |
index 3    | (sign extended to 16 bits)   |      | (sign extended to 16 bits)   |
           -------------------------------      -------------------------------
           |             ...              |      |             ...              |
           -------------------------------      -------------------------------
           |             ...              |      |             ...              |
           -15--------------------------0-      -15--------------------------0-
```

Attention

The iq order mode is only applied when tx/rx streaming is started and thus may not reflect the current iq order state.
If the iq order mode is set to skiq_iq_order_iq and an incompatible FPGA bitstream is then loaded via skiq_prog_fpga_from_file() or skiq_prog_fpga_from_flash(), the mode will automatically revert to skiq_iq_order_qi without warning.

Since

Function added in **v4.10.0**, requires FPGA **v3.12.0** or later

See Also

skiq_read_iq_order_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*mode*	[skiq_iq_order_t] skiq_iq_order_qi = use Q/I order mode (default) skiq_-iq_order_iq = use swapped order, I/Q

Returns

int32_t status where 0=success, anything else is an error

Return values

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-ENOSYS	if the FPGA version does not support IQ ordering mode
-ENOTSUP	if IQ order mode is not supported for the loaded FPGA bitstream
-EINVAL	if an invalid IQ order is specified. See skiq_iq_order_t

5.9.4.4 EPIQ_API int32_t skiq_read_iq_order_mode (uint8_t *card*, skiq_iq_order_t ∗ *p_mode*)

The skiq_read_iq_order_mode() function is responsible for retrieving the current I/Q order mode setting for the Sidekiq card.

Since

Function added in **v4.10.0**, requires FPGA **v3.12.0** or later

See Also

skiq_write_iq_order_mode

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_mode*	[skiq_iq_order_t] the currently set value of the order mode setting

Returns

int32_t status where 0=success, anything else is an error

Return values

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EFAULT	NULL pointer detected for `p_mode`
-EIO	A fault occurred communicating with the FPGA
-ENOSYS	FPGA does not meet minimum interface version requirements

5.9.4.5 EPIQ_API int32_t skiq_write_rx_data_src (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_data_src_t *src*)

The skiq_write_rx_data_src() function is responsible for setting the data source for the Rx interface. This is typically complex I/Q samples, but can also be set to use an incrementing counter for various test purposes. This must be set prior to calling skiq_start_rx_streaming() for the Rx interface.

Warning

If set after the Rx interface has been started, the setting will be stored but will not be used until streaming is stopped and re-started for the interface.

See Also

skiq_read_rx_data_src
skiq_receive
skiq_start_rx_streaming
skiq_start_rx_streaming_on_1pps

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested Rx interface
in	*src*	[skiq_data_src_t] the source of the data (either skiq_data_src_iq or skiq_data_src_counter)

Returns

int32_t status where 0=success, anything else is an error

5.9.4.6 EPIQ_API int32_t skiq_read_rx_data_src (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_data_src_t * *p_src*)

The skiq_read_rx_data_src() function is responsible for retrieving the currently set data source value (skiq_data_src_t).

See Also

skiq_write_rx_data_src

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] the handle of the requested Rx interface
out	*p_src*	[skiq_data_src_t] the currently set value of the pack mode setting

Returns

int32_t status where 0=success, anything else is an error

5.9.4.7 EPIQ_API int32_t skiq_read_fpga_semantic_version (uint8_t *card*, uint8_t * *p_major*, uint8_t * *p_minor*, uint8_t * *p_patch*)

The skiq_read_fpga_semantic_version() function is responsible for returning the major/minor/patch revision numbers for the currently loaded FPGA bitstream.

Since

Function added in API **v4.4.0**

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_major*	a pointer to where the major rev # should be returned
out	*p_minor*	a pointer to where the minor rev # should be returned
out	*p_patch*	a pointer to where the patch rev # should be returned

Returns

int32_t status where 0=success, anything else is an error

5.9.4.8 EPIQ_API int32_t skiq_read_fpga_tx_fifo_size (uint8_t *card*, skiq_fpga_tx_fifo_size_t * *p_tx_fifo_size*)

The skiq_read_fpga_tx_fifo_size() function is responsible for returning the Transmit FIFO size (skiq_fpga_tx_fifo_size_t representing the number of samples) for the currently loaded FPGA bitstream.

Since

Function added in API **v4.4.0**

Parameters

in	*card*	card index of the Sidekiq of interest

out	*p_tx_fifo_size*	[skiq_fpga_tx_fifo_size_t] reference to where the TX FIFO size enum should be returned

Returns

int32_t status where 0=success, anything else is an error

5.9.4.9 EPIQ_API int32_t skiq_write_user_fpga_reg (uint8_t *card*, uint32_t *addr*, uint32_t *data*)

The skiq_write_user_fpga_reg() function is used to update the 32-bit value of the requested user-definable FPGA register. This function is useful when adding custom logic to the FPGA, which can then controlled by software through this interface.

See Also

skiq_read_user_fpga_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*addr*	the register address to access in the FPGA's memory map
in	*data*	the 32-bit value to be written to the requested FPGA reg

Returns

int32_t status where 0=success, anything else is an error

5.9.4.10 EPIQ_API int32_t skiq_read_user_fpga_reg (uint8_t *card*, uint32_t *addr*, uint32_t * *p_data*)

The skiq_read_user_fpga_reg() function is responsible for reading out the current value in a user-definable FPGA register.

See Also

skiq_write_user_fpga_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*addr*	the register address to access in the FPGA's memory map
out	*p_data*	a pointer to a uint32_t to be updated with the current value of the requested FPGA register

Returns

int32_t status where 0=success, anything else is an error

5.9.4.11 EPIQ_API int32_t skiq_write_and_verify_user_fpga_reg (uint8_t *card*, uint32_t *addr*, uint32_t *data*)

The skiq_write_and_verify_user_fpga_reg() function is used to update the 32-bit value of the requested user-definable FPGA register. After the register has been written, this function verifies that reading the register returns the value previously written. This is useful to ensure that an FPGA register contains the expected value. This verification should be done in cases when performing a read immediately following the write since it is possible that the reads and writes could occur out-of-order, depending on the transport. Additionally, this is useful to verify in the cases where the register clock is running at a slower rate, such as the sample rate clock.

See Also

skiq_read_user_fpga_reg
skiq_write_user_fpga_reg

Since

Function added in API **v4.9.0**

Parameters

in	*card*	card index of the Sidekiq of interest
in	*addr*	the register address to access in the FPGA's memory map
in	*data*	the 32-bit value to be written to the requested FPGA reg

Return values

0	successful write and verification of user FPGA register
-EINVAL	specified card index is out of range
-EFAULT	addr is outside of valid FPGA user address range
-ENODEV	specified card index has not been initialized
-EIO	data readback does not match what was written

5.9.4.12 EPIQ_API int32_t skiq_prog_fpga_from_file (uint8_t *card*, FILE * *fp*)

The skiq_prog_fpga_from_file() function is responsible for programming the FPGA with an already opened bitstream file. This allows libsidekiq-based apps to reprogram the FPGA at run-time if needed.

Note

After successful reprogramming is complete, all RX interfaces are reset to the idle (not streaming) state.

Warning

Not all Sidekiq products support programming the FPGA from a file.

See Also

skiq_prog_fpga_from_flash
skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_from_flash
skiq_verify_fpga_config_in_flash_slot

Parameters

in	*card*	card index of the Sidekiq of interest
in	*fp*	pointer to already opened configuration file

Returns

0 on success, else a negative errno value

Return values

-ERANGE	The specified card index exceeds the maximum (SKIQ_MAX_NUM_CARDS)
-ENODEV	A card was not detected at the specified card index
-ENOTSUP	Configuring the FPGA from a file is not supported for this part
-EBADMSG	Error occurred transacting with FPGA registers
-EIO	Failed to configure the FPGA from the specified file pointer
-ESRCH	Internal error, Sidekiq transport misidentified or invalid
-ERANGE	Internal error, the system timestamp frequency indicated by the FPGA is out of range
-ENOTSUP	Internal error, Sidekiq RFIC does not support querying system timestamp frequency

5.9.4.13 EPIQ_API int32_t skiq_prog_fpga_from_flash (uint8_t *card*)

The skiq_prog_fpga_from_flash() function is responsible for programming the FPGA from the image previously stored in flash. This allows libsidekiq-based apps to reprogram the FPGA at run-time if needed.

Note

After successful reprogramming is complete, all RX interfaces are reset to the idle (not streaming) state.

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_from_flash
skiq_verify_fpga_config_in_flash_slot

Parameters

in	*card*	card index of the Sidekiq of interest

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index exceeds the maximum (SKIQ_MAX_NUM_CARDS)
-ENODEV	if a card was not detected at the specified card index
-EBADMSG	Error occurred transacting with FPGA registers
-EIO	Failed to configure the FPGA from the stored configuration bitstream
-ESRCH	Internal error, Sidekiq transport misidentified or invalid
-ERANGE	Internal error, the system timestamp frequency indicated by the FPGA is out of range

5.9.4.14 EPIQ_API int32_t skiq_save_fpga_config_to_flash (uint8_t *card*, FILE * *p_file*)

The skiq_save_fpga_config_to_flash() function stores a FPGA bitstream into flash memory, allowing it to be automatically loaded on power cycle or calling skiq_prog_fpga_from_flash().

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash
skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_from_flash
skiq_verify_fpga_config_in_flash_slot

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_file*	pointer to the FILE containing the FPGA bitstream

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EBADF	if the FILE stream references a bad file descriptor
-ENODEV	if no entry is found in the flash configuration array
-EACCES	if no golden FPGA bitstream is found in flash memory
-EIO	if the transport failed to read from flash memory
-EFAULT	if `p_file` is NULL
-ENOTSUP	if Flash access isn't supported for this card
-EFBIG	if the write would exceed Flash address boundaries and/or the flash config slot's size
-EFAULT	if the file specified by `p_file` doesn't contain an FPGA sync word
-ENOENT	(Internal Error) if the Flash data structure hasn't been initialized for this card

5.9.4.15 EPIQ_API int32_t skiq_verify_fpga_config_from_flash (uint8_t *card*, FILE * *p_file*)

The skiq_verify_fpga_config_from_flash() function verifies the contents of flash memory against a given file. This can be used to validate that a given FPGA bitstream is accurately stored within flash memory.

Since

Function added in API **v4.0.0**

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash
skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_in_flash_slot

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_file*	pointer to the FILE containing the FPGA bitstream to verify

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EFAULT	if `p_file` is NULL
-ENOTSUP	if Flash access isn't supported for this card
-EFBIG	if the file exceeds the Flash address boundaries
-EIO	if the file could not be read from
-EXDEV	if the verification failed
-ENOENT	(Internal Error) if the Flash data structure hasn't been initialized for this card

5.9.4.16 EPIQ_API int32_t skiq_read_golden_fpga_present_in_flash (uint8_t *card*, uint8_t * *p_present*)

The skiq_read_golden_fpga_present_in_flash() function is responsible for determining if there is a valid golden image stored in flash. The p_present is set based on whether a golden FPGA image is detected:

- 1 means the golden (fallback) FPGA is present
- 0 means the golden (fallback) FPGA is **NOT** present

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_present*	pointer to where to store an indication of whether the golden image is present

Returns

int32_t status where 0=success, anything else is an error

5.9.4.17 EPIQ_API int32_t skiq_read_last_tx_timestamp (uint8_t *card*, skiq_tx_hdl_t *hdl*, uint64_t ∗ *p_last_timestamp*)

The skiq_read_last_tx_timestamp() function queries the FPGA to determine what transmit timestamp it last encountered. The last transmit timestamp has two interpretations. Firstly, if the current RF timestamp is greater than the timestamp returned by this function, then the FPGA has already transmitted the block. Secondly, if the current RF timestamp is less than the timestamp returned by this function, then the FPGA is holding the transmit block and waiting until the RF timestamp matches the block's transmit timestamp.

Warning

The last transmit timestamp is only representative if the transmit flow mode is skiq_tx_with_timestamps_data_flow_mode.

Since

Function added in API **v4.0.0**, requires FPGA **v3.5** or later

See Also

skiq_read_tx_data_flow_mode
skiq_write_tx_data_flow_mode

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_tx_hdl_t] transmit handle of interest
out	*p_last_timestamp*	pointer to 64-bit timestamp value, will be zero if not transmitting

Returns

int32_t status where 0=success, anything else is an error

5.9.4.18 EPIQ_API int32_t skiq_prog_fpga_from_flash_slot (uint8_t *card*, uint8_t *slot*)

This function is responsible for programming the FPGA from an image stored in flash at the specified slot.

Note

A Sidekiq card can have anywhere between `1` and `N` slots available for storing FPGA images (bitstreams).
Use skiq_read_fpga_config_flash_slots_avail() to query the number of slots available.
The API function `skiq_prog_fpga_from_flash(card)` is equivalent to calling `skiq_prog_fpga_from_flash_slot(card, 0)`
After successful reprogramming is complete, all RX interfaces are reset to the idle (not streaming) state.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash
skiq_save_fpga_config_to_flash
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_from_flash
skiq_verify_fpga_config_in_flash_slot
skiq_read_fpga_config_flash_slot_metadata
skiq_find_fpga_config_flash_slot_metadata
skiq_read_fpga_config_flash_slots_avail

Parameters

in	*card*	requested Sidekiq card ID
in	*slot*	requested flash configuration slot

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EIO	if an error occurred during FPGA re-programming
-EBADMSG	if an error occurred transacting with FPGA registers
-ESRCH	(Internal Error) if transport cannot be resolved after programming

5.10 1PPS Functions and Definitions

These functions and definitions are related to interaction with the 1PPS pulse input of the Sidekiq SDR.

Enumerations

- enum skiq_1pps_source_t { skiq_1pps_source_unavailable =-1, skiq_1pps_source_external =0, skiq_1pps_source_host =1 }

  Source of 1PPS. Note that not all products support all configurations.

Functions

- EPIQ_API int32_t skiq_read_last_1pps_timestamp (uint8_t card, uint64_t ∗p_rf_timestamp, uint64_t ∗p_sys_timestamp)

  The skiq_read_last_1pps_timestamp() function is responsible for returning the RF and System timestamps of when the last 1PPS timestamp occurred.
- EPIQ_API int32_t skiq_write_timestamp_reset_on_1pps (uint8_t card, uint64_t future_sys_timestamp)
- EPIQ_API int32_t skiq_write_timestamp_update_on_1pps (uint8_t card, uint64_t future_sys_timestamp, uint64_t new_timestamp)
- EPIQ_API int32_t skiq_read_1pps_source (uint8_t card, skiq_1pps_source_t ∗p_pps_source)
- EPIQ_API int32_t skiq_write_1pps_source (uint8_t card, skiq_1pps_source_t pps_source)

5.10.1 Detailed Description

These functions and definitions are related to interaction with the 1PPS pulse input of the Sidekiq SDR.

5.10.2 Enumeration Type Documentation

5.10.2.1 enum skiq_1pps_source_t

Source of 1PPS. Note that not all products support all configurations.

See Also

skiq_read_1pps_source
skiq_write_1pps_source

Enumerator

skiq_1pps_source_unavailable

skiq_1pps_source_external 1PPS source from external connector

skiq_1pps_source_host 1PPS source from host connector

Definition at line 858 of file sidekiq_types.h.

5.10.3 Function Documentation

5.10.3.1 EPIQ_API int32_t skiq_read_last_1pps_timestamp (uint8_t *card*, uint64_t ∗ *p_rf_timestamp*, uint64_t ∗ *p_sys_timestamp*)

The skiq_read_last_1pps_timestamp() function is responsible for returning the RF and System timestamps of when the last 1PPS timestamp occurred.

Note

A user may pass *NULL* to `p_rf_timestamp` or `p_sys_timestamp` if the user is not interested in the value.

Attention

See Timestamp Slips within AD9361 Products for details on how calling this function can affect the RF timestamp metadata associated with received I/Q blocks.

Parameters

in	*card*	requested Sidekiq card ID
out	*p_rf_timestamp*	a uint64_t pointer where the value of the RF timestamp when the last 1PPS occurred, may be NULL
out	*p_sys_-timestamp*	a uint64_t pointer where the value of the System timestamp when the last 1PPS occurred, may be NULL

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EBADMSG	if an error occurred transacting with FPGA registers
-ERANGE	if timestamps could not be validated to be from the same 1PPS period

5.10.3.2 EPIQ_API int32_t skiq_write_timestamp_reset_on_1pps (uint8_t *card*, uint64_t *future_sys_timestamp*)

The skiq_write_timestamp_reset_on_1pps() function is responsible for configuring the FPGA to reset all the timestamps at a well defined point in the future. This point in the future is the occurrence of a 1PPS AFTER the specified system timestamp.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*future_sys_-timestamp*	the value of the system timestamp of a well defined point in the future, where the next 1PPS signal after this timestamp value will cause the timestamp to reset back to 0

Returns

int32_t status where 0=success, anything else is an error

5.10.3.3 EPIQ_API int32_t skiq_write_timestamp_update_on_1pps (uint8_t *card*, uint64_t *future_sys_timestamp*, uint64_t *new_timestamp*)

The skiq_write_timestamp_update_on_1pps() function is responsible for configuring the FPGA to set all timestamps to a specific value at a well defined point in the future. This point in the future is the occurrence of a 1PPS AFTER the specified system timestamp.

Parameters

in	*card*	card index of the Sidekiq of interest
in	*future_sys_timestamp*	the value of the system timestamp of a well defined point in the future, where the next 1PPS signal after this timestamp value will cause the timestamp to update to the value specified
in	*new_timestamp*	the value to set all timestamps to after the 1PPS

Returns

int32_t status where 0=success, anything else is an error

5.10.3.4 EPIQ_API int32_t skiq_read_1pps_source (uint8_t *card*, skiq_1pps_source_t * *p_pps_source*)

The skiq_read_1pps_source() function reads the currently configured source of the 1PPS signal.

Since

Function added in API **v4.7.0**

See Also

skiq_write_1pps_source

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_pps_source*	[skiq_1pps_source_t] pointer to 1pps source

Note

p_pps_source updated only upon success

Returns

int32_t

Return values

0	Success

-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EBADMSG	Error occurred transacting with FPGA registers
-ESRCH	Internal error, Sidekiq part misidentified or invalid

5.10.3.5 EPIQ_API int32_t skiq_write_1pps_source (uint8_t *card*, skiq_1pps_source_t *pps_source*)

The skiq_write_1pps_source() function configures the source of the 1PPS signal.

Note

Refer to the hardware user's manual for physical location of signal

Warning

Not all sources are available with all Sidekiq products

Attention

Supported sources may depend on FPGA bitstream

Since

Function added in API **v4.7.0**

See Also

skiq_read_1pps_source

Parameters

in	*card*	card index of the Sidekiq of interest
in	*pps_source*	[skiq_1pps_source_t] source of 1PPS signal

Returns

int32_t

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EBADMSG	Error occurred transacting with FPGA registers
-ENOSYS	FPGA bitstream does not support specified 1PPS source

-ENOTSUP	Sidekiq product does not specified 1PPS source
-EINVAL	Invalid 1PPS source specified

5.11 Crystal Oscillator (TCVCXO) Functions

These functions are related to configuration and usage of the on-board TCVCXO (Temperature Compensated / Voltage Controlled Crystal Oscillator) of the Sidekiq SDR.

Functions

- EPIQ_API int32_t skiq_write_tcvcxo_warp_voltage (uint8_t card, uint16_t warp_voltage)
- EPIQ_API int32_t skiq_read_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_read_default_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_read_user_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_write_user_tcvcxo_warp_voltage (uint8_t card, uint16_t warp_voltage)

5.11.1 Detailed Description

These functions are related to configuration and usage of the on-board TCVCXO (Temperature Compensated / Voltage Controlled Crystal Oscillator) of the Sidekiq SDR.

5.11.2 Function Documentation

5.11.2.1 EPIQ_API int32_t skiq_write_tcvcxo_warp_voltage (uint8_t *card,* uint16_t *warp_voltage*)

The skiq_write_tcvcxo_warp_voltage() function is responsible for setting a new warp value for the reference clock oscillator. A DAC is controlled by this function and the DAC can generate voltage between 0.75 and 2.-25V. Valid DAC values can vary from product to product, see product manual for details. Valid warp voltages for the ref clock oscillator are from 0.75 - 2.25V (which corresponds to evenly distributed values across all possible values in the DAC range).

See Also

skiq_read_tcvcxo_warp_voltage
skiq_read_default_tcvcxo_warp_voltage
skiq_read_user_tcvcxo_warp_voltage
skiq_write_user_tcvcxo_warp_voltage

Parameters

in	*card*	card index of the Sidekiq of interest
in	*warp_voltage*	a value corresponding to the desired DAC voltage to be applied. Valid values can vary from product to product, see product manual for details.

Returns

int32_t status where 0=success, anything else is an error

5.11.2.2 EPIQ_API int32_t skiq_read_tcvcxo_warp_voltage (uint8_t *card,* uint16_t * *p_warp_voltage*)

The skiq_read_txcvxo_warp_voltage() function is responsible for returning the current value of the warp voltage.

See Also

skiq_write_tcvcxo_warp_voltage
skiq_read_default_tcvcxo_warp_voltage
skiq_read_user_tcvcxo_warp_voltage
skiq_write_user_tcvcxo_warp_voltage

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_warp_voltage*	a pointer to where the currently set warp voltage will be written.

Returns

int32_t status where 0=success, anything else is an error

5.11.2.3 EPIQ_API int32_t skiq_read_default_tcvcxo_warp_voltage (uint8_t *card*, uint16_t * *p_warp_voltage*)

The skiq_read_default_txcvxo_warp_voltage() function is responsible for returning the default value of the warp voltage. This default value is determined during factory calibration and is read-only. If no factory calibrated value is available, an error is returned. The default TCVCXO warp voltage value is automatically loaded during skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str() unless a user value is defined in which case the user value is loaded during initialization.

See Also

skiq_write_tcvcxo_warp_voltage
skiq_read_tcvcxo_warp_voltage
skiq_read_user_tcvcxo_warp_voltage
skiq_write_user_tcvcxo_warp_voltage

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_warp_voltage*	a pointer to where the currently set warp voltage will be written.

Returns

int32_t status where 0=success, anything else is an error

5.11.2.4 EPIQ_API int32_t skiq_read_user_tcvcxo_warp_voltage (uint8_t *card*, uint16_t * *p_warp_voltage*)

The skiq_read_user_txcvxo_warp_voltage() function is responsible for returning the user defined warp voltage value. This value can be specified by the user and is automatically loaded during a call to skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str(). This value takes precedence over the default value loaded by the factory.

See Also

skiq_write_tcvcxo_warp_voltage
skiq_read_tcvcxo_warp_voltage
skiq_read_default_tcvcxo_warp_voltage
skiq_write_user_tcvcxo_warp_voltage

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_warp_voltage*	a pointer to where the currently set warp voltage will be written.

Returns

int32_t status where 0=success, anything else is an error

5.11.2.5 EPIQ_API int32_t skiq_write_user_tcvcxo_warp_voltage (uint8_t *card*, uint16_t *warp_voltage*)

The skiq_write_user_txcvxo_warp_voltage() function configures the user-defined warp voltage value. This value can be specified by the user and is automatically loaded during a call to skiq_init(), skiq_init_without_cards(), or skiq_init_by_serial_str(). This value takes precedence over the default value loaded by the factory.

See Also

skiq_write_tcvcxo_warp_voltage
skiq_read_tcvcxo_warp_voltage
skiq_read_default_tcvcxo_warp_voltage
skiq_read_user_tcvcxo_warp_voltage

Parameters

in	*card*	card index of the Sidekiq of interest
in	*warp_voltage*	specifies a warp voltage to set

Returns

int32_t status where 0=success, anything else is an error

5.12 Accelerometer Functions

These functions are related to using the Sidekiq's on-board accelerometer (`Analog Device's ADXL346`) for miniPCIe Sidekiq and M.2 Sidekiq products. The Sidekiq Z2, Sidekiq Stretch, and Matchstiq Z3u products use `TDK's InvenSense ICM-20602` motion tracking device. The accelerometer functions in this section are designed to function equivalently with their default configurations. Users may modify the behavior of the underlying device by using skiq_read_accel_reg() and skiq_write_accel_reg() as it suits their needs.

Functions

- EPIQ_API int32_t skiq_is_accel_supported (uint8_t card, bool ∗p_supported)
- EPIQ_API int32_t skiq_read_accel (uint8_t card, int16_t ∗p_x_data, int16_t ∗p_y_data, int16_t ∗p_z_data)
- EPIQ_API int32_t skiq_write_accel_state (uint8_t card, uint8_t enabled)
- EPIQ_API int32_t skiq_write_accel_reg (uint8_t card, uint8_t reg, uint8_t ∗p_data, uint32_t len)
- EPIQ_API int32_t skiq_read_accel_reg (uint8_t card, uint8_t reg, uint8_t ∗p_data, uint32_t len)
- EPIQ_API int32_t skiq_read_accel_state (uint8_t card, uint8_t ∗p_enabled)

5.12.1 Detailed Description

These functions are related to using the Sidekiq's on-board accelerometer (`Analog Device's ADXL346`) for miniPCIe Sidekiq and M.2 Sidekiq products. The Sidekiq Z2, Sidekiq Stretch, and Matchstiq Z3u products use `TDK's InvenSense ICM-20602` motion tracking device. The accelerometer functions in this section are designed to function equivalently with their default configurations. Users may modify the behavior of the underlying device by using skiq_read_accel_reg() and skiq_write_accel_reg() as it suits their needs.

5.12.2 Function Documentation

5.12.2.1 EPIQ_API int32_t skiq_is_accel_supported (uint8_t *card*, bool ∗ *p_supported*)

The skiq_is_accel_supported() function is responsible for determining if the accelerometer is supported on the hardware platform of the card specified.

Since

Function added in API **v4.2.0**

See Also

skiq_read_accel
skiq_read_accel_state
skiq_read_accel_reg
skiq_write_accel_state
skiq_write_accel_reg

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_supported*	pointer to where to accelerometer support

Returns

int32_t status where 0=success, anything else is an error

5.12.2.2 EPIQ_API int32_t skiq_read_accel (uint8_t *card*, int16_t ∗ *p_x_data*, int16_t ∗ *p_y_data*, int16_t ∗ *p_z_data*)

The skiq_read_accel() function is responsible for reading and providing the accelerometer data. The data format is twos compliment and 16 bits. If measurements are not available, -EAGAIN is returned and the accelerometer should be queried again for position.

Since

As of libsidekiq **v4.7.2**, for all supported products, this function will populate `p_x_data`, `p_y_data`, and `p_z_data` with measurements in units of thousandths of standard gravity (g_0).

See Also

skiq_is_accel_supported
skiq_read_accel_state
skiq_read_accel_reg
skiq_write_accel_state
skiq_write_accel_reg

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_x_data*	a pointer to where the X-axis accelerometer measurement is written
out	*p_y_data*	a pointer to where the Y-axis accelerometer measurement is written
out	*p_z_data*	a pointer to where the Z-axis accelerometer measurement is written

Returns

int32_t status where 0=success, anything else is an error

Return values

-ERANGE	specified card index is out of range
-ENODEV	specified card has not been initialized
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-EAGAIN	accelerometer measurement is not available
-EIO	error communicating with the accelerometer

5.12.2.3 EPIQ_API int32_t skiq_write_accel_state (uint8_t *card*, uint8_t *enabled*)

The skiq_write_accel_state() function is responsible for enabling or disabling the on-board accelerometer (if available) to take measurements.

See Also

skiq_is_accel_supported
skiq_read_accel
skiq_read_accel_state
skiq_read_accel_reg
skiq_write_accel_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*enabled*	accelerometer state (1=enabled, 0=disabled)

Returns

int32_t status where 0=success, anything else is an error

5.12.2.4 EPIQ_API int32_t skiq_write_accel_reg (uint8_t *card*, uint8_t *reg*, uint8_t ∗ *p_data*, uint32_t *len*)

The skiq_write_accel_reg() function provides generic write access to the on-board ADXL346 accelerometer.

Since

Function added in API **v4.2.0**

See Also

skiq_is_accel_supported
skiq_read_accel
skiq_read_accel_state
skiq_read_accel_reg
skiq_write_accel_state

Parameters

in	*card*	card index of the Sidekiq of interest
in	*reg*	register address to access
in	*p_data*	pointer to buffer of data to write
in	*len*	number of bytes to write

Returns

int32_t status where 0=success, anything else is an error

5.12.2.5 EPIQ_API int32_t skiq_read_accel_reg (uint8_t *card*, uint8_t *reg*, uint8_t ∗ *p_data*, uint32_t *len*)

The skiq_read_accel_reg() function provides generic read access to the onboard ADXL346 accelerometer.

Since

Function added in API **v4.2.0**

See Also

skiq_is_accel_supported
skiq_read_accel
skiq_read_accel_state
skiq_write_accel_state
skiq_write_accel_reg

Parameters

in	*card*	card index of the Sidekiq of interest
in	*reg*	register address to access
in	*p_data*	pointer to buffer to read data into
in	*len*	number of bytes to read

Returns

int32_t status where 0=success, anything else is an error

5.12.2.6 EPIQ_API int32_t skiq_read_accel_state (uint8_t *card*, uint8_t * *p_enabled*)

The skiq_read_accel_state() function is responsible for reading the current state of the accelerometer.

See Also

skiq_is_accel_supported
skiq_read_accel
skiq_read_accel_reg
skiq_write_accel_state
skiq_write_accel_reg

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_enabled*	pointer to where to store the accelerometer state (1=enabled, 0=disabled)

Returns

int32_t status where 0=success, anything else is an error

5.13 Receiver Calibration Functions

These functions and definitions are related to reading receiver calibration offsets (in dB) of the Sidekiq SDR.

Functions

- EPIQ_API int32_t skiq_read_rx_cal_offset (uint8_t card, skiq_rx_hdl_t hdl, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_gain_index (uint8_t card, skiq_rx_hdl_t hdl, uint8_t gain_index, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq_and_gain_index (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, uint8_t gain_index, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_data_present (uint8_t card, skiq_rx_hdl_t hdl, bool ∗p_present)
- EPIQ_API int32_t skiq_read_rx_cal_data_present_for_port (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t port, bool ∗p_present)
- EPIQ_API int32_t skiq_read_iq_complex_multiplier (uint8_t card, skiq_rx_hdl_t hdl, float_complex_t ∗p_factor)
- EPIQ_API int32_t skiq_write_iq_complex_multiplier_absolute (uint8_t card, skiq_rx_hdl_t hdl, float_complex_t factor)
- EPIQ_API int32_t skiq_write_iq_complex_multiplier_user (uint8_t card, skiq_rx_hdl_t hdl, float_complex_t factor)
- EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier (uint8_t card, skiq_rx_hdl_t hdl, float_complex_t ∗p_factor)
- EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier_by_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, float_complex_t ∗p_factor)
- EPIQ_API int32_t skiq_read_iq_complex_cal_data_present (uint8_t card, skiq_rx_hdl_t hdl, bool ∗p_present)

5.13.1 Detailed Description

These functions and definitions are related to reading receiver calibration offsets (in dB) of the Sidekiq SDR.

5.13.2 Function Documentation

5.13.2.1 EPIQ_API int32_t skiq_read_rx_cal_offset (uint8_t *card*, skiq_rx_hdl_t *hdl*, double ∗ *p_cal_off_dB*)

The skiq_read_rx_cal_offset() function provides a receive calibration offset based on the current settings of the receive handle. This function may not be used if the gain mode for the handle is set to skiq_rx_gain_auto and will return an error.

Since

Function added in API **v4.0.0**

See Also

skiq_read_rx_cal_offset_by_LO_freq
skiq_read_rx_cal_offset_by_gain_index
skiq_read_rx_cal_offset_by_LO_freq_and_gain_index
skiq_read_rx_cal_data_present
skiq_read_rx_cal_data_present_for_port

Parameters

	card	card index of the Sidekiq of interest
	hdl	[skiq_rx_hdl_t] receive handle of interest
out	*p_cal_off_dB*	reference to container for calibration offset in dB

Returns

int32_t status where 0=success, anything else is an error

5.13.2.2 EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *lo_freq*, double ∗ *p_cal_off_dB*)

The skiq_read_rx_cal_offset_by_LO_freq() function provides a receive calibration offset given an LO frequency and based on the present gain index of the receive handle. This function may not be used if the gain mode for the handle is set to skiq_rx_gain_auto and will return an error.

Since

Function added in API **v4.0.0**

See Also

skiq_read_rx_cal_offset
skiq_read_rx_cal_offset_by_gain_index
skiq_read_rx_cal_offset_by_LO_freq_and_gain_index
skiq_read_rx_cal_data_present
skiq_read_rx_cal_data_present_for_port

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*lo_freq*	LO frequency in Hertz
out	*p_cal_off_dB*	reference to container for calibration offset in dB

Returns

int32_t status where 0=success, anything else is an error

5.13.2.3 EPIQ_API int32_t skiq_read_rx_cal_offset_by_gain_index (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint8_t *gain_index*, double ∗ *p_cal_off_dB*)

The skiq_read_rx_cal_offset_by_gain_index() function provides a receive calibration offset given a receive gain index and based on the present LO frequency of the receive handle. This function is useful when the gain mode for the handle is set to skiq_rx_gain_auto and the caller feeds in the gain index from the receive packet'smetadata".

Since

Function added in API **v4.0.0**

See Also

skiq_read_rx_cal_offset
skiq_read_rx_cal_offset_by_LO_freq
skiq_read_rx_cal_offset_by_LO_freq_and_gain_index
skiq_read_rx_cal_data_present
skiq_read_rx_cal_data_present_for_port

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*gain_index*	gain index as set in the RFIC
out	*p_cal_off_dB*	reference to container for calibration offset in dB

Returns

int32_t status where 0=success, anything else is an error

5.13.2.4 EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq_and_gain_index (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *lo_freq*, uint8_t *gain_index*, double ∗ *p_cal_off_dB*)

The skiq_read_rx_cal_offset_by_LO_freq_and_gain_index() function provides a receive calibration offset given an LO frequency and receive gain index and based on the present RX FIR filter gain of the receive handle. This function is useful when the gain mode for the handle is set to skiq_rx_gain_auto and the caller feeds in the gain index from the receive packet's metadata and when the radio is not presently tuned to the frequency of interest.

Since

Function added in API **v4.0.0**

See Also

skiq_read_rx_cal_offset
skiq_read_rx_cal_offset_by_LO_freq
skiq_read_rx_cal_offset_by_gain_index
skiq_read_rx_cal_data_present
skiq_read_rx_cal_data_present_for_port

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest

in	*lo_freq*	LO frequency in Hertz
in	*gain_index*	gain index as set in the RFIC
out	*p_cal_off_dB*	reference to container for calibration offset in dB

Returns

int32_t status where 0=success, anything else is an error

5.13.2.5 EPIQ_API int32_t skiq_read_rx_cal_data_present (uint8_t *card*, skiq_rx_hdl_t *hdl*, bool ∗ *p_present*)

The skiq_read_rx_cal_data_present() function provides an indication for whether or not receiver calibration data is present for a specified card and handle. If the receiver calibration data is not present, the default calibration (if supported / available) in calibration offset queries.

Since

Function added in API **v4.4.0**

See Also

skiq_read_rx_cal_offset
skiq_read_rx_cal_offset_by_LO_freq
skiq_read_rx_cal_offset_by_gain_index
skiq_read_rx_cal_offset_by_LO_freq_and_gain_index
skiq_read_rx_cal_data_present_for_port

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_present*	reference to a boolean value indicating data presence

Returns

int32_t status where 0=success, anything else is an error

5.13.2.6 EPIQ_API int32_t skiq_read_rx_cal_data_present_for_port (uint8_t *card*, skiq_rx_hdl_t *hdl*, skiq_rf_port_t *port*, bool ∗ *p_present*)

The skiq_read_rx_cal_data_present_for_port() function provides an indication for whether or not receive calibration data is present for a specified card, handle, and RF port. If the receive calibration data is not present, the default calibration (if supported / available) is used in skiq_read_rx_cal_offset(), skiq_read_rx_cal_offset_by_LO_freq(), skiq_read_rx_cal_offset_by_gain_index(), and skiq_read_rx_cal_offset_by_LO_freq_and_gain_index().

Since

Function added in API **v4.5.0**

See Also

skiq_read_rx_cal_offset
skiq_read_rx_cal_offset_by_LO_freq
skiq_read_rx_cal_offset_by_gain_index
skiq_read_rx_cal_offset_by_LO_freq_and_gain_index
skiq_read_rx_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*port*	[skiq_rf_port_t] RF port of interest
out	*p_present*	reference to a boolean value indicating data presence

Returns

int32_t status where 0=success, anything else is an error

5.13.2.7 EPIQ_API int32_t skiq_read_iq_complex_multiplier (uint8_t *card,* skiq_rx_hdl_t *hdl,* float_complex_t ∗ *p_factor*)

The skiq_read_iq_complex_multiplier() function provides the complex multiplication factor that is currently in use for the supplied receive handle.

Attention

I/Q phase and amplitude multiplication factors are only supported on a subset of Sidekiq products and only if the FPGA is **v3.10.0** or later

Since

Function added in API **v4.7.0**, requires FPGA **v3.10.0** or later

See Also

skiq_read_iq_cal_complex_multiplier
skiq_read_iq_cal_complex_multiplier_by_LO_freq
skiq_write_iq_complex_multiplier_absolute
skiq_write_iq_complex_multiplier_user
skiq_read_iq_complex_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_factor*	[float_complex_t ∗] reference to the complex multiplication factor

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ENOSYS	Sidekiq platform is not running an FPGA that meets the minimum interface version requirements
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform
-EINVAL	An invalid / unsupported receive handle was specified

5.13.2.8 EPIQ_API int32_t skiq_write_iq_complex_multiplier_absolute (uint8_t *card*, skiq_rx_hdl_t *hdl*, float_complex_t *factor*)

The skiq_write_iq_complex_multiplier_absolute() function overwrites the complex multiplication factor that is currently in use for the supplied receive handle.

Attention

I/Q phase and amplitude multiplication factors are only supported on a subset of Sidekiq products and only if the FPGA is **v3.10.0** or later

Since

Function added in API **v4.7.0**, requires FPGA **v3.10.0** or later

See Also

skiq_read_iq_cal_complex_multiplier
skiq_read_iq_cal_complex_multiplier_by_LO_freq
skiq_read_iq_complex_multiplier
skiq_write_iq_complex_multiplier_user
skiq_read_iq_complex_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*factor*	[float_complex_t] complex multiplication factor to overwrite factory calibrated settings

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ENOSYS	Sidekiq platform is not running an FPGA that meets the minimum interface version requirements
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.13.2.9 EPIQ_API int32_t skiq_write_iq_complex_multiplier_user (uint8_t *card*, skiq_rx_hdl_t *hdl*, float_complex_t *factor*)

The skiq_write_iq_complex_multiplier_user() function further applies an I/Q phase and amplitude correction to the factory specified calibration factors. This function may be useful to users that have a two or four antenna configuration that they wish to "zero" out by applying an additional correction factor.

Attention

I/Q phase and amplitude multiplication factors are only supported on a subset of Sidekiq products and only if the FPGA is **v3.10.0** or later

```
i'[n] + j*q'[n] = (i[n] + j*q[n])*(re_cal + j*im_cal)*(re_user + j*im_user)
```

Since

Function added in API **v4.7.0**, requires FPGA **v3.10.0** or later

See Also

skiq_read_iq_cal_complex_multiplier
skiq_read_iq_cal_complex_multiplier_by_LO_freq
skiq_read_iq_complex_multiplier
skiq_write_iq_complex_multiplier_absolute
skiq_read_iq_complex_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*factor*	[float_complex_t] complex multiplication factor to apply in addition to factory calibrated settings

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ENOTSUP	Card index references a Sidekiq platform that does not currently support this functionality
-ENOSYS	Sidekiq platform is not running an FPGA that meets the minimum interface version requirements
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.13.2.10 EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier (uint8_t *card*, skiq_rx_hdl_t *hdl*, float_complex_t ∗ *p_factor*)

The skiq_read_iq_cal_complex_multiplier() function provides the complex multiplication factor based on the current settings of the receive handle as determined by factory settings.

Warning

The factors returned by this function may not represent the current factors in use whenever they are overwritten by skiq_write_iq_complex_multiplier_absolute() or skiq_write_iq_complex_multiplier_user(). Use the skiq_read_iq_complex_multiplier() instead to query the current factors.

Attention

IQ phase and amplitude calibration may be present but it is only active if the FPGA is **v3.10.0** or later.

Since

Function added in API **v4.7.0**

See Also

skiq_read_iq_cal_complex_multiplier_by_LO_freq
skiq_read_iq_complex_multiplier
skiq_write_iq_complex_multiplier_absolute
skiq_write_iq_complex_multiplier_user
skiq_read_iq_complex_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_factor*	[float_complex_t ∗] reference to the complex multiplication factor

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.13.2.11 EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier_by_LO_freq (uint8_t *card*, skiq_rx_hdl_t *hdl*, uint64_t *lo_freq*, float_complex_t ∗ *p_factor*)

The skiq_read_iq_cal_complex_multiplier_by_LO_freq() function provides the complex multiplication factor at given a receive LO frequency for the receive handle as determined by factory settings.

Warning

The factor returned by this function may not represent the current factor in use. They may have been overwritten by skiq_write_iq_complex_multiplier_absolute() or skiq_write_iq_complex_multiplier_user(). Use the skiq_read_iq_complex_multiplier() instead to query the factor that is currently in use.

Attention

IQ phase and amplitude calibration data may be present but is only active if the FPGA is **v3.10.0** or later.

Since

Function added in API **v4.7.0**

See Also

skiq_read_iq_cal_complex_multiplier
skiq_read_iq_complex_multiplier
skiq_write_iq_complex_multiplier_absolute
skiq_write_iq_complex_multiplier_user
skiq_read_iq_complex_cal_data_present

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
in	*lo_freq*	receive LO frequency of interest
out	*p_factor*	[float_complex_t ∗] reference to the complex multiplication factor

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.13.2.12 EPIQ_API int32_t skiq_read_iq_complex_cal_data_present (uint8_t *card,* skiq_rx_hdl_t *hdl,* bool ∗ *p_present*)

The skiq_read_iq_complex_cal_data_present() function provides an indication for whether or not I/Q phase and amplitude calibration data is present for a specified card and handle.

Warning

If the calibration data is not present, there is no default calibration. As such, there will be no IQ phase and amplitude correction.

Attention

I/Q phase and amplitude multiplication factors are only supported on a subset of Sidekiq products and only if the FPGA is **v3.10.0** or later

Since

Function added in API **v4.7.0**

See Also

skiq_read_iq_cal_complex_multiplier
skiq_read_iq_cal_complex_multiplier_by_LO_freq
skiq_read_iq_complex_multiplier
skiq_write_iq_complex_multiplier_absolute
skiq_write_iq_complex_multiplier_user

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	[skiq_rx_hdl_t] receive handle of interest
out	*p_present*	reference to a boolean value indicating data presence

Returns

int32_t status where 0=success, anything else is an error

Return values

0	Success
-ERANGE	Requested card index is out of range
-ENODEV	Requested card index is not initialized
-EDOM	Requested handle is not available or out of range for the Sidekiq platform

5.14 Flash Functions and Definitions

These functions and definitions are related to utilizing a Sidekiq's on-board flash storage for FPGA bit-stream(s).

Functions

- EPIQ_API int32_t skiq_save_fpga_config_to_flash_slot (uint8_t card, uint8_t slot, FILE *p_file, uint64_t metadata)

  This function stores a FPGA bitstream into flash memory at the specified slot. If the slot is `0`*, it is automatically loaded on power cycle or calling* `skiq_prog_fpga_from_flash(card)`*. If the slot is greater than* `0` *(and the card has more than one slot available), the FPGA configuration can be loaded by calling* `skiq_prog_fpga_from_flash_slot(card, slot)` *with the same specified* `slot` *value.*
- EPIQ_API int32_t skiq_verify_fpga_config_in_flash_slot (uint8_t card, uint8_t slot, FILE *p_file, uint64_t metadata)

  This function verifies the contents of flash memory at a specified against the provided FILE reference `p_file` *and* `metadata`*. This can be used to validate that a given FPGA bitstream and its metadata are accurately stored within flash memory.*
- EPIQ_API int32_t skiq_read_fpga_config_flash_slot_metadata (uint8_t card, uint8_t slot, uint64_t *p_metadata)

  This function reads the stored metadata associated with the specified slot value.
- EPIQ_API int32_t skiq_find_fpga_config_flash_slot_metadata (uint8_t card, uint64_t metadata, uint8_t *p_slot)

  This function uses calls to skiq_read_fpga_config_flash_slots_avail() and skiq_read_fpga_config_flash_slot_metadata() to provide the caller with the lowest slot index whose metadata matches the specified `metadata`*.*
- EPIQ_API int32_t skiq_read_fpga_config_flash_slots_avail (uint8_t card, uint8_t *p_nr_slots)

  This function provides the number of FPGA configuration slots available for a specified Sidekiq card.

5.14.1 Detailed Description

These functions and definitions are related to utilizing a Sidekiq's on-board flash storage for FPGA bit-stream(s).

5.14.2 Function Documentation

5.14.2.1 EPIQ_API int32_t skiq_save_fpga_config_to_flash_slot (uint8_t *card*, uint8_t *slot*, FILE * *p_file*, uint64_t *metadata*)

This function stores a FPGA bitstream into flash memory at the specified slot. If the slot is `0`, it is automatically loaded on power cycle or calling `skiq_prog_fpga_from_flash(card)`. If the slot is greater than `0` (and the card has more than one slot available), the FPGA configuration can be loaded by calling `skiq_prog_fpga_from_flash_slot(card, slot)` with the same specified `slot` value.

Note

A user may wish to store a hash or other related identifier of the bitstream in the `metadata` to make identifying the stored bitstream more robust than something another user may use (simple index or similar).
The specified `metadata` is stored with the FPGA configuration at the specified slot. This allows for a user to quickly associate the stored configuration among several images. This also then gives the user the

option to skip calling skiq_verify_fpga_config_in_flash_slot() since that function can take a relatively long time.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash
skiq_save_fpga_config_to_flash
skiq_verify_fpga_config_from_flash
skiq_verify_fpga_config_in_flash_slot
skiq_read_fpga_config_flash_slot_metadata
skiq_find_fpga_config_flash_slot_metadata
skiq_read_fpga_config_flash_slots_avail

Parameters

`in`	*card*	requested Sidekiq card ID
`in`	*slot*	requested flash configuration slot
`in`	*p_file*	FILE stream reference for the requested FPGA bitstream
`in`	*metadata*	metadata to associate with the FPGA bitstream at the specified slot

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EBADF	if the FILE stream references a bad file descriptor
-ENODEV	if no entry is found in the flash configuration array
-EACCES	if no golden FPGA bitstream is found in flash memory
-EIO	if the transport failed to read from flash memory
-EFAULT	if `p_file` is NULL
-ENOENT	if the Flash data structure hasn't been initialized for this card
-ENOTSUP	if Flash access isn't supported for this card
-EFBIG	if the write would exceed Flash address boundaries and/or the flash config slot's size
-EFAULT	if the file specified by `p_file` doesn't contain an FPGA sync word

5.14.2.2 EPIQ_API int32_t skiq_verify_fpga_config_in_flash_slot (uint8_t *card*, uint8_t *slot*, FILE ∗ *p_file*, uint64_t *metadata*)

This function verifies the contents of flash memory at a specified against the provided FILE reference `p_file` and `metadata`. This can be used to validate that a given FPGA bitstream and its metadata are accurately stored within flash memory.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_file
skiq_prog_fpga_from_flash
skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_from_flash
skiq_read_fpga_config_flash_slot_metadata
skiq_find_fpga_config_flash_slot_metadata
skiq_read_fpga_config_flash_slots_avail

Parameters

in	*card*	requested Sidekiq card ID
in	*slot*	requested flash configuration slot
in	*p_file*	FILE stream reference for the requested FPGA bitstream
in	*metadata*	metadata to verify at the specified slot

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-EBADF	if the FILE stream references a bad file descriptor
-EFBIG	if the FILE stream reference points to a file that exceeds the flash config slot's size
-EINVAL	if the `slot` index exceed number of accessible slots
-ENODEV	if no entry is found in the flash configuration array
-ENOTSUP	if Flash access isn't supported for this card
-EFAULT	if `p_file` is NULL
-ENOENT	(Internal Error) if the Flash data structure hasn't been initialized for this card

5.14.2.3 EPIQ_API int32_t skiq_read_fpga_config_flash_slot_metadata (uint8_t *card*, uint8_t *slot*, uint64_t * *p_metadata*)

This function reads the stored metadata associated with the specified slot value.

Note

This allows a user to be more efficient in determining which bitstreams are stored in a given Sidekiq card without having to dump the full contents of each flash slot.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_in_flash_slot
skiq_find_fpga_config_flash_slot_metadata
skiq_read_fpga_config_flash_slots_avail

Parameters

in	*card*	requested Sidekiq card ID
in	*slot*	requested flash configuration slot
out	*p_metadata*	populated with retrieved metadata when return value indicates success

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-ENODEV	if no entry is found in the flash configuration array
-EFAULT	if `p_metadata` is NULL
-EINVAL	if the `slot` index exceed number of accessible slots
-ENOENT	(Internal Error) if the Flash data structure hasn't been initialized for this card
-ENOTSUP	if Flash access isn't supported for this card
-EFBIG	(Internal Error) if the read would exceed Flash address boundaries

5.14.2.4 EPIQ_API int32_t skiq_find_fpga_config_flash_slot_metadata (uint8_t *card*, uint64_t *metadata*, uint8_t * *p_slot*)

This function uses calls to skiq_read_fpga_config_flash_slots_avail() and skiq_read_fpga_config_flash_slot_metadata() to provide the caller with the lowest slot index whose metadata matches the specified `metadata`.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_in_flash_slot
skiq_read_fpga_config_flash_slot_metadata
skiq_read_fpga_config_flash_slots_avail

Parameters

in	*card*	requested Sidekiq card ID
in	*metadata*	requested metadata
out	*p_slot*	populated with first slot index where metadata matches when return value indicates success

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-ENODEV	if no entry is found in the flash configuration array
-ENOENT	if the Flash data structure hasn't been initialized for this card
-ENOTSUP	if Flash access isn't supported for this card
-ESRCH	if the metadata was not found in any of the device's flash slots
-EFBIG	(Internal Error) if the read would exceed Flash address boundaries
-EFAULT	if `p_slot` is NULL

5.14.2.5 EPIQ_API int32_t skiq_read_fpga_config_flash_slots_avail (uint8_t *card,* uint8_t * *p_nr_slots*)

This function provides the number of FPGA configuration slots available for a specified Sidekiq card.

Note

A Sidekiq card can have anywhere between 0 and N slots available for storing FPGA images (bitstreams). See below for a caveat.

Warning

Some Sidekiq cards do not have slots that are accessible in every host or carrier configuration.

Since

Function added in API **v4.12.0**

See Also

skiq_prog_fpga_from_flash_slot
skiq_save_fpga_config_to_flash_slot
skiq_verify_fpga_config_in_flash_slot
skiq_read_fpga_config_flash_slot_metadata
skiq_find_fpga_config_flash_slot_metadata

Parameters

`in`	*card*	requested Sidekiq card ID
`out`	*p_nr_slots*	populated with the number of flash configuration slots when return value indicates success

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the requested card index is out of range
-ENODEV	if the requested card index is not initialized
-ENODEV	if no entry is found in the flash configuration array
-EFAULT	if `p_nr_slots` is NULL

5.15 GPS Disciplined Oscillator (GPSDO) Functions

These functions are related to status and availability of GPSDO for a product.

Enumerations

- enum skiq_gpsdo_support_t {
skiq_gpsdo_support_unknown = 0, skiq_gpsdo_support_is_supported, skiq_gpsdo_support_card_not_-supported, skiq_gpsdo_support_fpga_not_supported,
skiq_gpsdo_support_not_supported }
  The status of GPSDO support on a given card / FPGA bitstream.

Functions

- EPIQ_API int32_t skiq_is_gpsdo_supported (uint8_t card, skiq_gpsdo_support_t *p_supported)
  Indicates whether the GPSDO is available for product and FPGA bitstream.
- EPIQ_API int32_t skiq_gpsdo_enable (uint8_t card)
  Enable the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_gpsdo_disable (uint8_t card)
  Disable the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_gpsdo_is_enabled (uint8_t card, bool *p_is_enabled)
  Check the enable status of the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_gpsdo_read_freq_accuracy (uint8_t card, double *p_ppm)
  Calculate the frequency accuracy of the FPGA's GPSDO oscillator frequency (in ppm)
- EPIQ_API int32_t skiq_gpsdo_is_locked (uint8_t card, bool *p_is_locked)
  Check the lock status of the GPSDO control algorithm on the specified card.

5.15.1 Detailed Description

These functions are related to status and availability of GPSDO for a product.

5.15.2 Enumeration Type Documentation

5.15.2.1 enum skiq_gpsdo_support_t

The status of GPSDO support on a given card / FPGA bitstream.

Since

Definition added in **v4.15.0**

Enumerator

skiq_gpsdo_support_unknown The GPSDO support is unknown or cannot be read for this card.

skiq_gpsdo_support_is_supported The card and FPGA bitstream support GPSDO functionality.

skiq_gpsdo_support_card_not_supported The Sidekiq card does not support GPSDO functionality.

skiq_gpsdo_support_fpga_not_supported The loaded FPGA bitstream does not support GPSDO functionality.

skiq_gpsdo_support_not_supported The card and/or FPGA bitstream are capable of GPSDO functionality but have indicated that it is not currently supported.

Definition at line 973 of file sidekiq_types.h.

5.15.3 Function Documentation

5.15.3.1 EPIQ_API int32_t skiq_is_gpsdo_supported (uint8_t *card*, skiq_gpsdo_support_t * *p_supported*)

Indicates whether the GPSDO is available for product and FPGA bitstream.

Since

Function added in API **v4.15.0**

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_supported*	the status of the GPSDO support on the specified card

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized
-EFAULT	if `p_supported` is NULL
-EBADMSG	if an error occurred transacting with FPGA registers

5.15.3.2 EPIQ_API int32_t skiq_gpsdo_enable (uint8_t *card*)

Enable the GPSDO control algorithm on the specified card.

Attention

When the GPSDO is enabled, the FPGA takes control of the warp voltage thus disabling manual control of the voltage. Specifically, **skiq_write_tcvcxo_warp_voltage()** is not allowed when GPSDO enabled. When GPSDO is enabled, the FPGA takes ownership of the temperature sensor. Temperature data may not immediately be available, as noted by the -EAGAIN error code returned when the temperature is queried via **skiq_read_temp()**

Since

Function added in API **v4.15.0**

Parameters

`in`	*card*	card index of the Sidekiq of interest

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized
-ENOTSUP	if the specified card does not support an FPGA-based GPSDO
-ENOSYS	if the loaded FPGA bitstream does not implement GPSDO functionality
-EBADMSG	if an error occurred transacting with FPGA registers

5.15.3.3 EPIQ_API int32_t skiq_gpsdo_disable (uint8_t *card*)

Disable the GPSDO control algorithm on the specified card.

Since

Function added in API **v4.15.0**

Parameters

`in`	*card*	card index of the Sidekiq of interest

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized
-ENOTSUP	if the specified card does not support an FPGA-based GPSDO
-ENOSYS	if the loaded FPGA bitstream does not implement GPSDO functionality
-EBADMSG	if an error occurred transacting with FPGA registers

5.15.3.4 EPIQ_API int32_t skiq_gpsdo_is_enabled (uint8_t *card,* bool ∗ *p_is_enabled*)

Check the enable status of the GPSDO control algorithm on the specified card.

Since

Function added in API **v4.15.0**

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_is_enabled*	true if the GPSDO algorithm is enabled, else false

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized
-EFAULT	if `p_is_enabled` is NULL
-ENOTSUP	if the specified card does not support an FPGA-based GPSDO
-ENOSYS	if the loaded FPGA bitstream does not implement GPSDO functionality
-EBADMSG	if an error occurred transacting with FPGA registers

5.15.3.5 EPIQ_API int32_t skiq_gpsdo_read_freq_accuracy (uint8_t *card*, double * *p_ppm*)

Calculate the frequency accuracy of the FPGA's GPSDO oscillator frequency (in ppm)

Note

The developer may also want to use the skiq_gpsdo_is_locked() API function if skiq_gpsdo_read_freq_accuracy() returns *-EAGAIN* to determine what condition is causing the function to indicate failure

Since

Function added in API **v4.15.0**

See Also

skiq_gpsdo_is_locked

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_ppm*	calculated ppm (parts per million) of the GPSDO's frequency accuracy

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized

-ENOTSUP	if the specified card does not support an FPGA-based GPSDO
-ENOSYS	if the loaded FPGA bitstream does not implement GPSDO functionality
-ESRCH	if the measurement is not available because the GPSDO is disabled
-EAGAIN	if the measurement is not available because: • the GPS module does not have a valid fix -OR- • the GPSDO algorithm is not locked
-EBADMSG	if an error occurred transacting with FPGA registers
-EFAULT	if NULL is provided for p_ppm

5.15.3.6 EPIQ_API int32_t skiq_gpsdo_is_locked (uint8_t *card*, bool ∗ *p_is_locked*)

Check the lock status of the GPSDO control algorithm on the specified card.

Since

Function added in API **v4.17.0**

Parameters

`in`	*card*	card index of the Sidekiq of interest
`out`	*p_is_locked*	true if the GPSDO is locked, else false

Returns

0 on success, else a negative errno value

Return values

-ERANGE	if the specified card index is out of range
-ENODEV	if the specified card has not been initialized
-ENOTSUP	if the specified card does not support an FPGA-based GPSDO
-ENOSYS	if the loaded FPGA bitstream does not implement GPSDO functionality
-EBADMSG	if an error occurred transacting with FPGA registers
-EFAULT	if NULL is provided for p_is_locked

Chapter 6

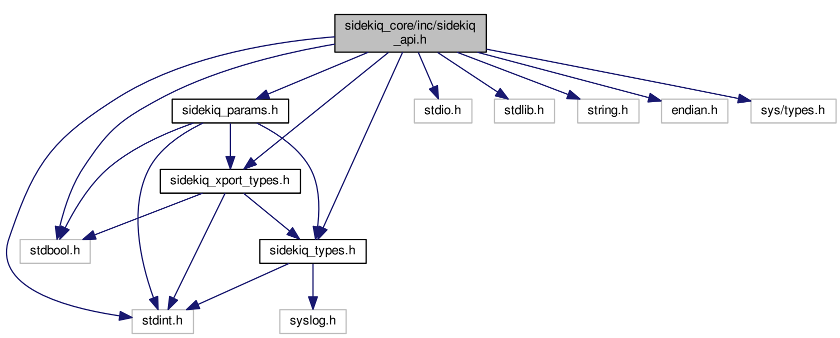
File Documentation

6.1 sidekiq_core/inc/sidekiq_api.h File Reference

This file contains the public interface of the sidekiq_api provided by libsidekiq.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <endian.h>
#include <sys/types.h>
#include "sidekiq_types.h"
#include "sidekiq_params.h"
#include "sidekiq_xport_types.h"
```

Include dependency graph for sidekiq_api.h:



Macros

- #define LIBSIDEKIQ_VERSION_MAJOR 4

  Major version number for libsidekiq.
- #define LIBSIDEKIQ_VERSION_MINOR 18

Minor version number for libsidekiq.
- #define LIBSIDEKIQ_VERSION_PATCH 0

  Patch version number for libsidekiq.
- #define LIBSIDEKIQ_VERSION_LABEL ""

  Label version for libsidekiq.
- #define LIBSIDEKIQ_VERSION

  Version of libsidekiq. e.g., To test for LIBSIDEKIQ_VERSION > 3.6.1.
- #define SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS 1024

  SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS is the largest block size that can be transferred between the FPGA and the CPU in a single transaction when receiving.
- #define SKIQ_MAX_RX_BLOCK_SIZE_IN_BYTES

  The same parameter as SKIQ_MAX_RX_BLOCK_SIZE_IN_WORDS except calculated in bytes.
- #define SKIQ_RX_HEADER_SIZE_IN_WORDS 6

  The current Rx header size is 6 words but may change in the future. The metadata placed at the beginning of each IQ block. Refer to skiq_receive() for details on the formatting of the metadata.
- #define SKIQ_RX_HEADER_SIZE_IN_BYTES

  The current Rx header size, only in bytes.
- #define SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK(block_size_in_words)

  When running in packed mode, every 4 samples are 3 words of data. SKIQ_NUM_PACKED_SAMPLES_IN_BLOCK converts from number of words to number of samples when running in packed mode.
- #define SKIQ_NUM_WORDS_IN_PACKED_BLOCK(num_packed_samples)

  When running in packed mode, every 3 words of data contain 4 samples. SKIQ_NUM_WORDS_IN_PACKED_BLO-CK converts from the number of samples to the number of words needed to hold the number of unpacked samples. The SKIQ_NUM_WORDS_IN_PACKED_BLOCK macro rounds up by adding one less than the denominator (the number of bytes in a word: 4) prior to performing the integer division.
- #define SKIQ_RX_NUM_PACKETS_IN_RING_BUFFER (2048)

  The number of packets in the ring buffer is the number of packets that can be buffered and not yet received prior to the packets getting overwritten.
- #define SKIQ_MAX_TX_PACKET_SIZE_IN_WORDS 65536

  The largest number of words that can be transferred between the FPGA and the CPU. This includes both the data block as well as the header size.
- #define SKIQ_TX_HEADER_SIZE_IN_WORDS 4

  The current Tx header size is fixed at 4 words of metadata for now at the start of each I/Q block, which may well increase at some point. For details on the exact format and contents of the transmit packet, refer to skiq_transmit()
- #define SKIQ_TX_TIMESTAMP_OFFSET_IN_WORDS 2

  The offset (in 32-bit words) to the header where the Tx timestamp is stored.
- #define SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS

  SKIQ_MAX_TX_BLOCK_SIZE_IN_WORDS is the largest block size of sample data that can be transferred from the CPU to the FPGA while transmitting. Note that a "block" of data includes the sample data minus the header data
- #define SKIQ_TX_HEADER_SIZE_IN_BYTES

  The current Tx header size, only in bytes.
- #define SKIQ_TX_PACKET_SIZE_INCREMENT_IN_WORDS (256)

  The Tx packet must be in increments of 256 words. Note: the packet size accounts for both the header size as well as the block (sample) size.
- #define SKIQ_START_USER_FPGA_REG_ADDR 0x00008700

  SKIQ_START_USER_FPGA_REG_ADDR is first address available in the FPGA memory map that can be user defined. These 32-bit register addresses increment by 4 bytes
- #define SKIQ_END_USER_FPGA_REG_ADDR 0x00008FFF

  SKIQ_END_USER_FPGA_REG_ADDR is last address of the last FPGA register available in the FPGA memory map that can be user defined.

- #define SKIQ_MIN_LO_FREQ 47000000LLU

  SKIQ_MIN_LO_FREQ defines the minimum acceptable RF frequency for the Rx/Tx LO for a standard Sidekiq.
- #define SKIQ_MAX_LO_FREQ 6000000000LLU

  SKIQ_MAX_LO_FREQ defines the maximum acceptable RF frequency for the Rx/Tx LO for a standard Sidekiq.
- #define SKIQ_MIN_SAMPLE_RATE 233000

  SKIQ_MIN_SAMPLE_RATE is the minimum Rx/Tx sample rate that can be generated for a single Rx/Tx channel.
- #define SKIQ_MAX_SAMPLE_RATE 122880000

  SKIQ_MAX_SAMPLE_RATE is the maximum Rx/Tx sample rate that can be generated for a single Rx/Tx channel.
- #define SKIQ_MAX_DUAL_CHAN_MPCIE_SAMPLE_RATE 30720000

  SKIQ_MAX_DUAL_CHAN_MPCIE_SAMPLE_RATE is the maximum sample rate that can be generated when running in dual channel mode on a Sidekiq mPCIe (skiq_mpcie) product. Note: this rate can be extended higher, but only with certain caveats, so this is kept at a reasonably safe value for all use cases by default.
- #define SKIQ_MAX_DUAL_CHAN_Z3U_SAMPLE_RATE 30720000

  SKIQ_MAX_DUAL_CHAN_Z3U_SAMPLE_RATE is the maximum sample rate that can be generated when running in dual channel mode on a Matchstiq Z3u (skiq_z3u) product.
- #define SKIQ_MAX_TX_ATTENUATION (359)

  SKIQ_MAX_TX_ATTENUATION is the maximum value of the Tx attenuation.
- #define SKIQ_MIN_RX_GAIN (0)

  SKIQ_MIN_RX_GAIN is the minimum value of the Rx gain.
- #define SKIQ_MAX_RX_GAIN (76)

  SKIQ_MAX_RX_GAIN is the maximum value of the Rx gain.
- #define SKIQ_MAX_NUM_CARDS (32)

  SKIQ_MAX_NUM_CARDS is the maximum number of Sidekiq cards that is supported in a system
- #define SKIQ_SYS_TIMESTAMP_FREQ (40000000)

  SKIQ_SYS_TIMESTAMP_FREQ is the frequency at which the system timestamp increments
- #define SKIQ_RX_SYS_META_WORD_OFFSET (4)

  SKIQ_RX_SYS_META_WORD_OFFSET is the offset at which the system metadata is located within a receive packet. Included in this is the Rx handle as well as the overload bit
- #define SKIQ_RX_USER_META_WORD_OFFSET (5)

  SKIQ_RX_USER_META_WORD_OFFSET is the offset at which the user-defined metadata is located with a receive packet
- #define SKIQ_RX_META_HDL_BITS (0x3F)

  SKIQ_RX_META_HDL_BITS is the bitmask which represent the Rx handle
- #define SKIQ_RX_META_OVERLOAD_BIT (1 << 6)

  SKIQ_RX_META_OVERLOAD_BIT is the location of the bit representing the Rx overload detection in the miscellaneous metadata of a receive packet
- #define SKIQ_RX_META_RFIC_CTRL_BITS (0xFF)

  SKIQ_RX_META_RFIC_CTRL_BITS are the bits which contain the RFIC control bits embedded within the system metadata
- #define SKIQ_RX_META_RFIC_CTRL_OFFSET (7)

  SKIQ_RX_META_RFIC_CTRL_OFFSET is the bit offset where the RFIC control bits are located within the system metadata
- #define SKIQ_TX_ASYNC_SEND_QUEUE_FULL (100)

  SKIQ_TX_ASYNC_SEND_QUEUE_FULL is the return code of the skiq_transmit() call when using skiq_tx_transfer_mode_async and there is no space available to store the data to send
- #define SKIQ_TX_MAX_NUM_THREADS (10)

  SKIQ_TX_MAX_NUM_THREADS is the maximum number of threads used in transmitting when using skiq_tx_transfer_mode_async
- #define SKIQ_TX_MIN_NUM_THREADS (2)

SKIQ_TX_MIN_NUM_THREADS is the minimum number of threads used in transmitting when skiq_tx_transfer_-mode_async

- #define RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA1 (0x16)

  RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA1 is the value that should be used to enable the gain values for RxA1 to be presented in the system metadata of each receive packet. Use this definition in conjunction with skiq_write_rfic_control_output_config()
- #define RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA2 (0x17)

  RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA2 is the value that should be used to enable the gain values for RxA2 to be presented in the system metadata of each receive packet. Use this definition in conjunction with skiq_write_rfic_control_output_config()
- #define RFIC_CONTROL_OUTPUT_MODE_GAIN_BITS (0x7F)

  RFIC_CONTROL_OUTPUT_MODE_GAIN_BITS are the bits used in conveying the current gain setting (read from the RFIC control output). Use this definition in conjunction with skiq_write_rfic_control_output_config()
- #define RX_TRANSFER_NO_WAIT (0)

  Option for timeout_us argument of skiq_set_rx_transfer_timeout() to return immediately, regardless as to whether or not samples are available. Effectively results in a non-blocking skiq_receive() call and the return code is set accordingly.
- #define RX_TRANSFER_WAIT_FOREVER (-1)

  Option for timeout_us argument of skiq_set_rx_transfer_timeout() to block forever until samples are available. Effectively results in a blocking skiq_receive() call with no timeout. Use with caution (or don't use at all)
- #define RX_TRANSFER_WAIT_NOT_SUPPORTED (-2)

  Possible value for ***p_timeout_us*** *argument of skiq_get_rx_transfer_timeout() to indicate that blocking skiq_-receive() is not supported by the card and/or its currently configured transport layer (skiq_xport_type_t).*

Functions

- static void skiq_tx_set_block_timestamp (skiq_tx_block_t *p_block, uint64_t timestamp)
- static uint64_t skiq_tx_get_block_timestamp (skiq_tx_block_t *p_block)
- static skiq_tx_block_t * skiq_tx_block_allocate_by_bytes (uint32_t data_size_in_bytes)
- static skiq_tx_block_t * skiq_tx_block_allocate (uint32_t data_size_in_samples)
- static void skiq_tx_block_free (skiq_tx_block_t *p_block)
- EPIQ_API int32_t skiq_get_cards (skiq_xport_type_t xport_type, uint8_t *p_num_cards, uint8_t *p_-cards)
- EPIQ_API int32_t skiq_read_serial_string (uint8_t card, char **pp_serial_num)
- EPIQ_API int32_t skiq_get_card_from_serial_string (char *p_serial_num, uint8_t *p_card)
- EPIQ_API int32_t skiq_init (skiq_xport_type_t type, skiq_xport_init_level_t level, uint8_t *p_card_-nums, uint8_t num_cards)
- EPIQ_API int32_t skiq_enable_cards (const uint8_t cards[ ], uint8_t num_cards, skiq_xport_init_level_t level)
- EPIQ_API int32_t skiq_enable_cards_by_serial_str (const char **pp_serial_nums, uint8_t num_cards, skiq_xport_init_level_t level, uint8_t *p_card_nums)
- EPIQ_API int32_t skiq_init_by_serial_str (skiq_xport_type_t type, skiq_xport_init_level_t level, char **pp_serial_nums, uint8_t num_cards, uint8_t *p_card_nums)
- EPIQ_API int32_t skiq_init_without_cards (void)
- EPIQ_API int32_t skiq_read_parameters (uint8_t card, skiq_param_t *p_param)
- EPIQ_API int32_t skiq_is_xport_avail (uint8_t card, skiq_xport_type_t type)
- EPIQ_API int32_t skiq_is_card_avail (uint8_t card, pid_t *p_card_owner)
- EPIQ_API int32_t skiq_exit (void)
- EPIQ_API int32_t skiq_disable_cards (const uint8_t cards[ ], uint8_t num_cards)
- EPIQ_API int32_t skiq_read_rx_streaming_handles (uint8_t card, skiq_rx_hdl_t *p_hdls_streaming, uint8_t *p_num_hdls)

- EPIQ_API int32_t skiq_read_rx_stream_handle_conflict (uint8_t card, skiq_rx_hdl_t hdl_to_stream, skiq_rx_hdl_t ∗p_conflicting_hdls, uint8_t ∗p_num_hdls)
- EPIQ_API int32_t skiq_start_rx_streaming (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_immediate (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_synced (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_start_rx_streaming_on_1pps (uint8_t card, skiq_rx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_start_rx_streaming_multi_on_trigger (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, skiq_trigger_src_t trigger, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_start_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_start_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_stop_rx_streaming (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_immediate (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_synced (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles)
- EPIQ_API int32_t skiq_stop_rx_streaming_on_1pps (uint8_t card, skiq_rx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_stop_rx_streaming_multi_on_trigger (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, skiq_trigger_src_t trigger, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_stop_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_stop_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
- EPIQ_API int32_t skiq_read_last_1pps_timestamp (uint8_t card, uint64_t ∗p_rf_timestamp, uint64_t ∗p_sys_timestamp)

  The skiq_read_last_1pps_timestamp() function is responsible for returning the RF and System timestamps of when the last 1PPS timestamp occurred.
- EPIQ_API int32_t skiq_write_timestamp_reset_on_1pps (uint8_t card, uint64_t future_sys_timestamp)
- EPIQ_API int32_t skiq_write_timestamp_update_on_1pps (uint8_t card, uint64_t future_sys_timestamp, uint64_t new_timestamp)
- EPIQ_API int32_t skiq_read_tx_timestamp_base (uint8_t card, skiq_tx_timestamp_base_t ∗p_timestamp_base)
- EPIQ_API int32_t skiq_write_tx_timestamp_base (uint8_t card, skiq_tx_timestamp_base_t timestamp_base)
- EPIQ_API int32_t skiq_read_tx_data_flow_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_flow_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_tx_data_flow_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_flow_mode_t mode)
- EPIQ_API int32_t skiq_read_tx_transfer_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_transfer_mode_t ∗p_transfer_mode)
- EPIQ_API int32_t skiq_write_tx_transfer_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_transfer_mode_t transfer_mode)
- EPIQ_API int32_t skiq_register_tx_complete_callback (uint8_t card, skiq_tx_callback_t tx_complete)
- EPIQ_API int32_t skiq_register_tx_enabled_callback (uint8_t card, skiq_tx_ena_callback_t tx_ena_cb)
- EPIQ_API int32_t skiq_read_chan_mode (uint8_t card, skiq_chan_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_chan_mode (uint8_t card, skiq_chan_mode_t mode)
- EPIQ_API int32_t skiq_write_rx_preselect_filter_path (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t path)
- EPIQ_API int32_t skiq_read_rx_preselect_filter_path (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t ∗p_path)

- EPIQ_API int32_t skiq_write_tx_filter_path (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t path)
- EPIQ_API int32_t skiq_read_tx_filter_path (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t *p_path)
- EPIQ_API int32_t skiq_read_rx_overload_state (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *p_overload)
- EPIQ_API int32_t skiq_read_rx_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t *p_freq, double *p_actual_freq)
- EPIQ_API int32_t skiq_write_rx_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t freq)
- EPIQ_API int32_t skiq_read_rx_sample_rate (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_rate, double *p_actual_rate)
- EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t rate, uint32_t bandwidth)
- EPIQ_API int32_t skiq_write_rx_sample_rate_and_bandwidth_multi (uint8_t card, skiq_rx_hdl_t handles[ ], uint8_t nr_handles, uint32_t rate[ ], uint32_t bandwidth[ ])
- EPIQ_API int32_t skiq_read_rx_sample_rate_and_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_rate, double *p_actual_rate, uint32_t *p_bandwidth, uint32_t *p_actual_bandwidth)
- EPIQ_API int32_t skiq_write_tx_sample_rate_and_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t rate, uint32_t bandwidth)
- EPIQ_API int32_t skiq_read_tx_sample_rate_and_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t *p_rate, double *p_actual_rate, uint32_t *p_bandwidth, uint32_t *p_actual_bandwidth)
- EPIQ_API int32_t skiq_set_rx_transfer_timeout (const uint8_t card, const int32_t timeout_us)
- EPIQ_API int32_t skiq_get_rx_transfer_timeout (const uint8_t card, int32_t *p_timeout_us)
- EPIQ_API skiq_rx_status_t skiq_receive (uint8_t card, skiq_rx_hdl_t *p_hdl, skiq_rx_block_t **pp_block, uint32_t *p_data_len)
- EPIQ_API int32_t skiq_transmit (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_block_t *p_block, void *p_user)
- EPIQ_API int32_t skiq_read_rx_gain_index_range (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *p_gain_index_min, uint8_t *p_gain_index_max)
- EPIQ_API int32_t skiq_write_rx_gain (uint8_t card, skiq_rx_hdl_t hdl, uint8_t gain_index)
- EPIQ_API int32_t skiq_read_rx_gain (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *p_gain_index)
- EPIQ_API int32_t skiq_read_rx_gain_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_gain_t *p_gain_mode)
- EPIQ_API int32_t skiq_write_rx_gain_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_gain_t gain_mode)
- EPIQ_API int32_t skiq_write_rx_attenuation_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_attenuation_mode_t mode)
- EPIQ_API int32_t skiq_read_rx_attenuation_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_attenuation_mode_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_attenuation (uint8_t card, skiq_rx_hdl_t hdl, uint16_t attenuation)
- EPIQ_API int32_t skiq_read_rx_attenuation (uint8_t card, skiq_rx_hdl_t hdl, uint16_t *p_attenuation)
- EPIQ_API int32_t skiq_read_tx_LO_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t *p_freq, double *p_tuned_freq)
- EPIQ_API int32_t skiq_write_tx_LO_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t freq)
- EPIQ_API int32_t skiq_enable_tx_tone (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_disable_tx_tone (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_tx_tone_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t *p_freq)
- EPIQ_API int32_t skiq_read_tx_tone_freq_offset (uint8_t card, skiq_tx_hdl_t hdl, int32_t *p_freq_offset)
- EPIQ_API int32_t skiq_write_tx_tone_freq_offset (uint8_t card, skiq_tx_hdl_t hdl, int32_t test_freq_offset)
- EPIQ_API int32_t skiq_write_tx_attenuation (uint8_t card, skiq_tx_hdl_t hdl, uint16_t attenuation)
- EPIQ_API int32_t skiq_read_tx_attenuation (uint8_t card, skiq_tx_hdl_t hdl, uint16_t *p_attenuation)
- EPIQ_API int32_t skiq_read_tx_sample_rate (uint8_t card, skiq_tx_hdl_t hdl, uint32_t *p_rate, double *p_actual_rate)

- EPIQ_API int32_t skiq_read_tx_block_size (uint8_t card, skiq_tx_hdl_t hdl, uint16_t *p_block_size_in_words)
- EPIQ_API int32_t skiq_write_tx_block_size (uint8_t card, skiq_tx_hdl_t hdl, uint16_t block_size_in_words)
- EPIQ_API int32_t skiq_read_tx_num_underruns (uint8_t card, skiq_tx_hdl_t hdl, uint32_t *p_num_underrun)
- EPIQ_API int32_t skiq_read_tx_num_late_timestamps (uint8_t card, skiq_tx_hdl_t hdl, uint32_t *p_num_late)
- EPIQ_API int32_t skiq_read_temp (uint8_t card, int8_t *p_temp_in_deg_C)
- EPIQ_API int32_t skiq_is_accel_supported (uint8_t card, bool *p_supported)
- EPIQ_API int32_t skiq_read_accel (uint8_t card, int16_t *p_x_data, int16_t *p_y_data, int16_t *p_z_data)
- EPIQ_API int32_t skiq_write_accel_state (uint8_t card, uint8_t enabled)
- EPIQ_API int32_t skiq_write_accel_reg (uint8_t card, uint8_t reg, uint8_t *p_data, uint32_t len)
- EPIQ_API int32_t skiq_read_accel_reg (uint8_t card, uint8_t reg, uint8_t *p_data, uint32_t len)
- EPIQ_API int32_t skiq_read_accel_state (uint8_t card, uint8_t *p_enabled)
- EPIQ_API int32_t skiq_write_tcvcxo_warp_voltage (uint8_t card, uint16_t warp_voltage)
- EPIQ_API int32_t skiq_read_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_read_default_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_read_user_tcvcxo_warp_voltage (uint8_t card, uint16_t *p_warp_voltage)
- EPIQ_API int32_t skiq_write_user_tcvcxo_warp_voltage (uint8_t card, uint16_t warp_voltage)
- EPIQ_API int32_t skiq_write_iq_pack_mode (uint8_t card, bool mode)
- EPIQ_API int32_t skiq_read_iq_pack_mode (uint8_t card, bool *p_mode)
- EPIQ_API int32_t skiq_write_iq_order_mode (uint8_t card, skiq_iq_order_t mode)
- EPIQ_API int32_t skiq_read_iq_order_mode (uint8_t card, skiq_iq_order_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_data_src (uint8_t card, skiq_rx_hdl_t hdl, skiq_data_src_t src)
- EPIQ_API int32_t skiq_read_rx_data_src (uint8_t card, skiq_rx_hdl_t hdl, skiq_data_src_t *p_src)
- EPIQ_API int32_t skiq_write_rx_stream_mode (uint8_t card, skiq_rx_stream_mode_t stream_mode)
- EPIQ_API int32_t skiq_read_rx_stream_mode (uint8_t card, skiq_rx_stream_mode_t *p_stream_mode)
- EPIQ_API int32_t skiq_read_curr_rx_timestamp (uint8_t card, skiq_rx_hdl_t hdl, uint64_t *p_timestamp)
- EPIQ_API int32_t skiq_read_curr_tx_timestamp (uint8_t card, skiq_tx_hdl_t hdl, uint64_t *p_timestamp)
- EPIQ_API int32_t skiq_read_curr_sys_timestamp (uint8_t card, uint64_t *p_timestamp)
- EPIQ_API int32_t skiq_reset_timestamps (uint8_t card)
- EPIQ_API int32_t skiq_update_timestamps (uint8_t card, uint64_t new_timestamp)
- EPIQ_API int32_t skiq_read_libsidekiq_version (uint8_t *p_major, uint8_t *p_minor, uint8_t *p_patch, const char **p_label)
- EPIQ_API int32_t skiq_read_fpga_version (uint8_t card, uint32_t *p_git_hash, uint32_t *p_build_date, uint8_t *p_major, uint8_t *p_minor, skiq_fpga_tx_fifo_size_t *p_tx_fifo_size)
- EPIQ_API int32_t skiq_read_fpga_semantic_version (uint8_t card, uint8_t *p_major, uint8_t *p_minor, uint8_t *p_patch)
- EPIQ_API int32_t skiq_read_fpga_tx_fifo_size (uint8_t card, skiq_fpga_tx_fifo_size_t *p_tx_fifo_size)
- EPIQ_API int32_t skiq_read_fw_version (uint8_t card, uint8_t *p_major, uint8_t *p_minor)
- EPIQ_API int32_t skiq_read_hw_version (uint8_t card, skiq_hw_vers_t *p_hw_version)
- EPIQ_API int32_t skiq_read_product_version (uint8_t card, skiq_product_t *p_product)
- EPIQ_API int32_t skiq_write_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t data)
- EPIQ_API int32_t skiq_read_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t *p_data)
- EPIQ_API int32_t skiq_write_and_verify_user_fpga_reg (uint8_t card, uint32_t addr, uint32_t data)
- EPIQ_API int32_t skiq_prog_rfic_from_file (FILE *fp, uint8_t card)
- EPIQ_API int32_t skiq_prog_fpga_from_file (uint8_t card, FILE *fp)

- EPIQ_API int32_t skiq_prog_fpga_from_flash (uint8_t card)
- EPIQ_API int32_t skiq_save_fpga_config_to_flash (uint8_t card, FILE *p_file)
- EPIQ_API int32_t skiq_verify_fpga_config_from_flash (uint8_t card, FILE *p_file)
- EPIQ_API const char * skiq_part_string (skiq_part_t part)
- EPIQ_API const char * skiq_hardware_vers_string (skiq_hw_vers_t hardware_vers)
- EPIQ_API const char * skiq_product_vers_string (skiq_product_t product_vers)
- EPIQ_API const char * skiq_rf_port_string (skiq_rf_port_t rf_port)
- EPIQ_API int32_t skiq_read_part_info (uint8_t card, char *p_part_number, char *p_revision, char *p_variant)
- EPIQ_API int32_t skiq_read_max_sample_rate (uint8_t card, uint32_t *p_max_sample_rate)
- EPIQ_API int32_t skiq_read_min_sample_rate (uint8_t card, uint32_t *p_min_sample_rate)
- EPIQ_API int32_t skiq_write_rx_dc_offset_corr (uint8_t card, skiq_rx_hdl_t hdl, bool enable)
- EPIQ_API int32_t skiq_read_rx_dc_offset_corr (uint8_t card, skiq_rx_hdl_t hdl, bool *p_enable)
- EPIQ_API int32_t skiq_read_rfic_reg (uint8_t card, uint16_t addr, uint8_t *p_data)
- EPIQ_API int32_t skiq_write_rfic_reg (uint8_t card, uint16_t addr, uint8_t data)
- EPIQ_API int32_t skiq_read_rfic_tx_fir_config (uint8_t card, uint8_t *p_num_taps, uint8_t *p_fir_interpolation)
- EPIQ_API int32_t skiq_read_rfic_tx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_write_rfic_tx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_read_rfic_rx_fir_config (uint8_t card, uint8_t *p_num_taps, uint8_t *p_fir_decimation)
- EPIQ_API int32_t skiq_read_rfic_rx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_write_rfic_rx_fir_coeffs (uint8_t card, int16_t *p_coeffs)
- EPIQ_API int32_t skiq_write_rx_fir_gain (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_fir_gain_t gain)
- EPIQ_API int32_t skiq_read_rx_fir_gain (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_fir_gain_t *p_gain)
- EPIQ_API int32_t skiq_write_tx_fir_gain (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_fir_gain_t gain)
- EPIQ_API int32_t skiq_read_tx_fir_gain (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_fir_gain_t *p_gain)
- EPIQ_API int32_t skiq_read_ref_clock_select (uint8_t card, skiq_ref_clock_select_t *p_ref_clk)
- EPIQ_API int32_t skiq_read_ext_ref_clock_freq (uint8_t card, uint32_t *p_freq)
- EPIQ_API int32_t skiq_read_num_tx_threads (uint8_t card, uint8_t *p_num_threads)
- EPIQ_API int32_t skiq_write_num_tx_threads (uint8_t card, uint8_t num_threads)
- EPIQ_API int32_t skiq_read_tx_thread_priority (uint8_t card, int32_t *p_priority)
- EPIQ_API int32_t skiq_write_tx_thread_priority (uint8_t card, int32_t priority)
- EPIQ_API void skiq_register_critical_error_callback (void(*critical_handler)(int32_t status, void *p_user_data), void *p_user_data)
- EPIQ_API void skiq_register_logging (void(*log_msg)(int32_t priority, const char *message))
- EPIQ_API int32_t skiq_read_num_rx_chans (uint8_t card, uint8_t *p_num_rx_chans)
- EPIQ_API int32_t skiq_read_num_tx_chans (uint8_t card, uint8_t *p_num_tx_chans)
- EPIQ_API int32_t skiq_read_rx_iq_resolution (uint8_t card, uint8_t *p_adc_res)
- EPIQ_API int32_t skiq_read_tx_iq_resolution (uint8_t card, uint8_t *p_dac_res)
- EPIQ_API int32_t skiq_read_golden_fpga_present_in_flash (uint8_t card, uint8_t *p_present)
- EPIQ_API int32_t skiq_read_rfic_control_output_rx_gain_config (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *p_mode, uint8_t *p_ena)
- EPIQ_API int32_t skiq_write_rfic_control_output_config (uint8_t card, uint8_t mode, uint8_t ena)
- EPIQ_API int32_t skiq_read_rfic_control_output_config (uint8_t card, uint8_t *p_mode, uint8_t *p_ena)
- EPIQ_API int32_t skiq_enable_rfic_control_output_rx_gain (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_rx_LO_freq_range (uint8_t card, uint64_t *p_max, uint64_t *p_min)
- EPIQ_API int32_t skiq_read_max_rx_LO_freq (uint8_t card, uint64_t *p_max)
- EPIQ_API int32_t skiq_read_min_rx_LO_freq (uint8_t card, uint64_t *p_min)
- EPIQ_API int32_t skiq_read_tx_LO_freq_range (uint8_t card, uint64_t *p_max, uint64_t *p_min)

- EPIQ_API int32_t skiq_read_max_tx_LO_freq (uint8_t card, uint64_t ∗p_max)
- EPIQ_API int32_t skiq_read_min_tx_LO_freq (uint8_t card, uint64_t ∗p_min)
- EPIQ_API int32_t skiq_read_rx_filters_avail (uint8_t card, skiq_rx_hdl_t hdl, skiq_filt_t ∗p_filters, uint8_t ∗p_num_filters)
- EPIQ_API int32_t skiq_read_tx_filters_avail (uint8_t card, skiq_tx_hdl_t hdl, skiq_filt_t ∗p_filters, uint8_t ∗p_num_filters)
- EPIQ_API int32_t skiq_read_filter_range (skiq_filt_t filter, uint64_t ∗p_start_freq, uint64_t ∗p_end_freq)
- EPIQ_API int32_t skiq_read_rf_port_config_avail (uint8_t card, bool ∗p_fixed, bool ∗p_trx)
- EPIQ_API int32_t skiq_read_rf_port_config (uint8_t card, skiq_rf_port_config_t ∗p_config)
- EPIQ_API int32_t skiq_write_rf_port_config (uint8_t card, skiq_rf_port_config_t config)
- EPIQ_API int32_t skiq_read_rf_port_operation (uint8_t card, bool ∗p_transmit)
- EPIQ_API int32_t skiq_write_rf_port_operation (uint8_t card, bool transmit)
- EPIQ_API int32_t skiq_read_rx_rf_ports_avail_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, uint8_t ∗p_num_fixed_rf_ports, skiq_rf_port_t ∗p_fixed_rf_port_list, uint8_t ∗p_num_trx_rf_ports, skiq_rf_port_t ∗p_trx_rf_port_list)
- EPIQ_API int32_t skiq_read_rx_rf_port_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t ∗p_rf_port)
- EPIQ_API int32_t skiq_write_rx_rf_port_for_hdl (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t rf_port)
- EPIQ_API int32_t skiq_read_tx_rf_ports_avail_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, uint8_t ∗p_num_fixed_rf_ports, skiq_rf_port_t ∗p_fixed_rf_port_list, uint8_t ∗p_num_trx_rf_ports, skiq_rf_port_t ∗p_trx_rf_port_list)
- EPIQ_API int32_t skiq_read_tx_rf_port_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, skiq_rf_port_t ∗p_rf_port)
- EPIQ_API int32_t skiq_write_tx_rf_port_for_hdl (uint8_t card, skiq_tx_hdl_t hdl, skiq_rf_port_t rf_port)
- EPIQ_API int32_t skiq_read_rx_cal_offset (uint8_t card, skiq_rx_hdl_t hdl, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_gain_index (uint8_t card, skiq_rx_hdl_t hdl, uint8_t gain_index, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_offset_by_LO_freq_and_gain_index (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, uint8_t gain_index, double ∗p_cal_off_dB)
- EPIQ_API int32_t skiq_read_rx_cal_data_present (uint8_t card, skiq_rx_hdl_t hdl, bool ∗p_present)
- EPIQ_API int32_t skiq_read_rx_cal_data_present_for_port (uint8_t card, skiq_rx_hdl_t hdl, skiq_rf_port_t port, bool ∗p_present)
- EPIQ_API int32_t skiq_read_last_tx_timestamp (uint8_t card, skiq_tx_hdl_t hdl, uint64_t ∗p_last_timestamp)
- EPIQ_API int32_t skiq_read_usb_enumeration_delay (uint8_t card, uint16_t ∗p_delay_ms)
- EPIQ_API int32_t skiq_read_sys_timestamp_freq (uint8_t card, uint64_t ∗p_sys_timestamp_freq)
- EPIQ_API int32_t skiq_read_rx_block_size (uint8_t card, skiq_rx_stream_mode_t stream_mode)
- EPIQ_API int32_t skiq_read_tx_quadcal_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_quadcal_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_tx_quadcal_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_quadcal_mode_t mode)
- EPIQ_API int32_t skiq_run_tx_quadcal (uint8_t card, skiq_tx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_rx_cal_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_cal_mode_t ∗p_mode)
- EPIQ_API int32_t skiq_write_rx_cal_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rx_cal_mode_t mode)
- EPIQ_API int32_t skiq_run_rx_cal (uint8_t card, skiq_rx_hdl_t hdl)
- EPIQ_API int32_t skiq_read_rx_cal_type_mask (uint8_t card, skiq_rx_hdl_t hdl, uint32_t ∗p_cal_mask)

- EPIQ_API int32_t skiq_write_rx_cal_type_mask (uint8_t card, skiq_rx_hdl_t hdl, uint32_t cal_mask)
- EPIQ_API int32_t skiq_read_rx_cal_types_avail (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p_cal_mask)
- EPIQ_API int32_t skiq_read_iq_complex_multiplier (uint8_t card, skiq_rx_hdl_t hdl, float_complex_t *p_factor)
- EPIQ_API int32_t skiq_write_iq_complex_multiplier_absolute (uint8_t card, skiq_rx_hdl_t hdl, float_-complex_t factor)
- EPIQ_API int32_t skiq_write_iq_complex_multiplier_user (uint8_t card, skiq_rx_hdl_t hdl, float_-complex_t factor)
- EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier (uint8_t card, skiq_rx_hdl_t hdl, float_-complex_t *p_factor)
- EPIQ_API int32_t skiq_read_iq_cal_complex_multiplier_by_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t lo_freq, float_complex_t *p_factor)
- EPIQ_API int32_t skiq_read_iq_complex_cal_data_present (uint8_t card, skiq_rx_hdl_t hdl, bool *p_-present)
- EPIQ_API int32_t skiq_read_1pps_source (uint8_t card, skiq_1pps_source_t *p_pps_source)
- EPIQ_API int32_t skiq_write_1pps_source (uint8_t card, skiq_1pps_source_t pps_source)
- EPIQ_API int32_t skiq_read_calibration_date (uint8_t card, uint16_t *p_last_cal_year, uint8_t *p_last-_cal_week, uint8_t *p_cal_interval)
- EPIQ_API int32_t skiq_write_rx_freq_tune_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_freq_tune_-mode_t mode)
- EPIQ_API int32_t skiq_read_rx_freq_tune_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_freq_tune_-mode_t *p_mode)
- EPIQ_API int32_t skiq_write_tx_freq_tune_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_freq_tune_-mode_t mode)
- EPIQ_API int32_t skiq_read_tx_freq_tune_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_freq_tune_-mode_t *p_mode)
- EPIQ_API int32_t skiq_write_rx_freq_hop_list (uint8_t card, skiq_rx_hdl_t hdl, uint16_t num_freq, uint64_t freq_list[ ], uint16_t initial_index)
- EPIQ_API int32_t skiq_read_rx_freq_hop_list (uint8_t card, skiq_rx_hdl_t hdl, uint16_t *p_num_freq, uint64_t freq_list[SKIQ_MAX_NUM_FREQ_HOPS])
- EPIQ_API int32_t skiq_write_tx_freq_hop_list (uint8_t card, skiq_tx_hdl_t hdl, uint16_t num_freq, uint64_t freq_list[ ], uint16_t initial_index)
- EPIQ_API int32_t skiq_read_tx_freq_hop_list (uint8_t card, skiq_tx_hdl_t hdl, uint16_t *p_num_freq, uint64_t freq_list[SKIQ_MAX_NUM_FREQ_HOPS])
- EPIQ_API int32_t skiq_write_next_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t freq_index)
- EPIQ_API int32_t skiq_write_next_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t freq_index)
- EPIQ_API int32_t skiq_perform_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint64_t rf_timestamp)
- EPIQ_API int32_t skiq_perform_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint64_t rf_timestamp)
- EPIQ_API int32_t skiq_read_curr_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t *p_hop_index, uint64_t *p_curr_freq)
- EPIQ_API int32_t skiq_read_next_rx_freq_hop (uint8_t card, skiq_rx_hdl_t hdl, uint16_t *p_hop_-index, uint64_t *p_curr_freq)
- EPIQ_API int32_t skiq_read_curr_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t *p_hop_index, uint64_t *p_curr_freq)
- EPIQ_API int32_t skiq_read_next_tx_freq_hop (uint8_t card, skiq_tx_hdl_t hdl, uint16_t *p_hop_index, uint64_t *p_curr_freq)
- EPIQ_API int32_t skiq_prog_fpga_from_flash_slot (uint8_t card, uint8_t slot)

  This function is responsible for programming the FPGA from an image stored in flash at the specified slot.
- EPIQ_API int32_t skiq_save_fpga_config_to_flash_slot (uint8_t card, uint8_t slot, FILE *p_file, uint64_t metadata)

This function stores a FPGA bitstream into flash memory at the specified slot. If the slot is `0`*, it is automatically loaded on power cycle or calling* `skiq_prog_fpga_from_flash(card)`*. If the slot is greater than* `0` *(and the card has more than one slot available), the FPGA configuration can be loaded by calling* `skiq_prog_fpga_from_flash_slot(card, slot)` *with the same specified* `slot` *value.*

- EPIQ_API int32_t skiq_verify_fpga_config_in_flash_slot (uint8_t card, uint8_t slot, FILE *p_file, uint64_t metadata)

  This function verifies the contents of flash memory at a specified against the provided FILE reference `p_file` *and* `metadata`*. This can be used to validate that a given FPGA bitstream and its metadata are accurately stored within flash memory.*
- EPIQ_API int32_t skiq_read_fpga_config_flash_slot_metadata (uint8_t card, uint8_t slot, uint64_t *p_metadata)

  This function reads the stored metadata associated with the specified slot value.
- EPIQ_API int32_t skiq_find_fpga_config_flash_slot_metadata (uint8_t card, uint64_t metadata, uint8_t *p_slot)

  This function uses calls to skiq_read_fpga_config_flash_slots_avail() and skiq_read_fpga_config_flash_slot_metadata() to provide the caller with the lowest slot index whose metadata matches the specified `metadata`*.*
- EPIQ_API int32_t skiq_read_fpga_config_flash_slots_avail (uint8_t card, uint8_t *p_nr_slots)

  This function provides the number of FPGA configuration slots available for a specified Sidekiq card.
- EPIQ_API int32_t skiq_set_exit_handler_state (bool enabled)

  Set the state of the exit handler.
- EPIQ_API int32_t skiq_write_ref_clock_select (uint8_t card, skiq_ref_clock_select_t ref_clock_source)

  This function allows the user to switch between different reference clock sources. This change is run-time only and is not written to the card nor permanent.
- EPIQ_API int32_t skiq_write_ext_ref_clock_freq (uint8_t card, uint32_t ext_freq)

  This function allows the user to switch between different external reference clock frequencies. This change is run-time only and is not written to the card nor permanent. This will automatically update the reference clock selection to an external reference clock source. When changing the frequency, a supported external reference clock frequency must be used per the card specification.
- EPIQ_API int32_t skiq_write_rx_rfic_pin_ctrl_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rfic_pin_mode_t mode)

  skiq_write_rx_rfic_pin_ctrl_mode selects the source of RFIC Rx enable on supported RFICs. This signal disables or enables the receiver signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.
- EPIQ_API int32_t skiq_write_tx_rfic_pin_ctrl_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_rfic_pin_mode_t mode)

  skiq_write_tx_rfic_pin_ctrl_mode selects the source of RFIC Tx enable on supported RFICs. This signal disables or enables the transmitter signal path. Normally managed in software by libsidekiq, some Sidekiq platforms can be controlled by the FPGA.
- EPIQ_API int32_t skiq_read_rx_rfic_pin_ctrl_mode (uint8_t card, skiq_rx_hdl_t hdl, skiq_rfic_pin_mode_t *p_mode)

  This function reads the source of control used to enable/disable RFIC Rx.
- EPIQ_API int32_t skiq_read_tx_rfic_pin_ctrl_mode (uint8_t card, skiq_tx_hdl_t hdl, skiq_rfic_pin_mode_t *p_mode)

  This function reads the source of control used to enable/disable RFIC Tx.
- EPIQ_API int32_t skiq_is_gpsdo_supported (uint8_t card, skiq_gpsdo_support_t *p_supported)

  Indicates whether the GPSDO is available for product and FPGA bitstream.
- EPIQ_API int32_t skiq_gpsdo_enable (uint8_t card)

  Enable the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_gpsdo_disable (uint8_t card)

  Disable the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_gpsdo_is_enabled (uint8_t card, bool *p_is_enabled)

Check the enable status of the GPSDO control algorithm on the specified card.

- EPIQ_API int32_t skiq_gpsdo_read_freq_accuracy (uint8_t card, double *p_ppm)

  Calculate the frequency accuracy of the FPGA's GPSDO oscillator frequency (in ppm)
- EPIQ_API int32_t skiq_gpsdo_is_locked (uint8_t card, bool *p_is_locked)

  Check the lock status of the GPSDO control algorithm on the specified card.
- EPIQ_API int32_t skiq_read_rx_analog_filter_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t *p-_bandwidth)
- EPIQ_API int32_t skiq_read_tx_analog_filter_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t *p-_bandwidth)
- EPIQ_API int32_t skiq_write_rx_analog_filter_bandwidth (uint8_t card, skiq_rx_hdl_t hdl, uint32_t bandwidth)
- EPIQ_API int32_t skiq_write_tx_analog_filter_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t bandwidth)
- EPIQ_API int32_t skiq_write_rx_sample_shift (uint8_t card, skiq_rx_hdl_t hdl, uint8_t shift_delay)
- EPIQ_API int32_t skiq_read_rx_sample_shift (uint8_t card, skiq_rx_hdl_t hdl, uint8_t *shift_delay)

6.1.1 Detailed Description

This file contains the public interface of the sidekiq_api provided by libsidekiq.

Definition in file sidekiq_api.h.

6.1.2 Macro Definition Documentation

6.1.2.1 #define LIBSIDEKIQ_VERSION_MAJOR 4

Major version number for libsidekiq.

Definition at line 314 of file sidekiq_api.h.

6.1.2.2 #define LIBSIDEKIQ_VERSION_MINOR 18

Minor version number for libsidekiq.

Definition at line 317 of file sidekiq_api.h.

6.1.2.3 #define LIBSIDEKIQ_VERSION_PATCH 0

Patch version number for libsidekiq.

Definition at line 320 of file sidekiq_api.h.

6.1.2.4 #define LIBSIDEKIQ_VERSION_LABEL ""

Label version for libsidekiq.

Definition at line 326 of file sidekiq_api.h.

6.1.2.5 #define LIBSIDEKIQ_VERSION

Value:

```
(LIBSIDEKIQ_VERSION_MAJOR * 10000 \
                            + LIBSIDEKIQ_VERSION_MINOR * 100 \
                            + LIBSIDEKIQ_VERSION_PATCH)
```

Version of libsidekiq. e.g., To test for LIBSIDEKIQ_VERSION > 3.6.1.

```
#if LIBSIDEKIQ_VERSION > 30601
```

Definition at line 336 of file sidekiq_api.h.

6.1.2.6 #define SKIQ_MIN_LO_FREQ 47000000LLU

SKIQ_MIN_LO_FREQ defines the minimum acceptable RF frequency for the Rx/Tx LO for a standard Sidekiq.

Deprecated To determine the min LO frequency use skiq_read_rx_LO_freq_range() or skiq_read_min_rx_LO_freq().

Definition at line 463 of file sidekiq_api.h.

6.1.2.7 #define SKIQ_MAX_LO_FREQ 6000000000LLU

SKIQ_MAX_LO_FREQ defines the maximum acceptable RF frequency for the Rx/Tx LO for a standard Sidekiq.

Deprecated To determine the max LO frequency use skiq_read_rx_LO_freq_range() or skiq_read_max_rx_LO_freq().

Definition at line 470 of file sidekiq_api.h.

6.1.2.8 #define SKIQ_MIN_SAMPLE_RATE 233000

SKIQ_MIN_SAMPLE_RATE is the minimum Rx/Tx sample rate that can be generated for a single Rx/Tx channel.

Deprecated To determine the minimum sample rate for the specific hardware / radio configuration, refer to skiq_read_parameters.

Definition at line 477 of file sidekiq_api.h.

6.1.2.9 #define SKIQ_MAX_SAMPLE_RATE 122880000

SKIQ_MAX_SAMPLE_RATE is the maximum Rx/Tx sample rate that can be generated for a single Rx/Tx channel.

Note

this rate can be extended higher, but only with certain caveats, so this is kept at a reasonably safe value for all use cases by default.

Deprecated To determine the maximum sample rate for the specific hardware / radio configuration, refer to skiq_read_parameters.

Definition at line 487 of file sidekiq_api.h.

6.1.2.10 #define SKIQ_MAX_DUAL_CHAN_MPCIE_SAMPLE_RATE 30720000

SKIQ_MAX_DUAL_CHAN_MPCIE_SAMPLE_RATE is the maximum sample rate that can be generated when running in dual channel mode on a Sidekiq mPCIe (skiq_mpcie) product. Note: this rate can be extended higher, but only with certain caveats, so this is kept at a reasonably safe value for all use cases by default.

Definition at line 494 of file sidekiq_api.h.

6.1.2.11 #define SKIQ_MAX_DUAL_CHAN_Z3U_SAMPLE_RATE 30720000

SKIQ_MAX_DUAL_CHAN_Z3U_SAMPLE_RATE is the maximum sample rate that can be generated when running in dual channel mode on a Matchstiq Z3u (skiq_z3u) product.

Definition at line 499 of file sidekiq_api.h.

6.1.2.12 #define SKIQ_MAX_TX_ATTENUATION (359)

SKIQ_MAX_TX_ATTENUATION is the maximum value of the Tx attenuation.

Deprecated Use skiq_read_parameters() and the corresponding skiq_param_t struct to determine the attenuation range.

Definition at line 505 of file sidekiq_api.h.

6.1.2.13 #define SKIQ_MIN_RX_GAIN (0)

SKIQ_MIN_RX_GAIN is the minimum value of the Rx gain.

Deprecated To determine the minimum Rx gain, use skiq_read_rx_gain_index_range().

Definition at line 511 of file sidekiq_api.h.

6.1.2.14 #define SKIQ_MAX_RX_GAIN (76)

SKIQ_MAX_RX_GAIN is the maximum value of the Rx gain.

Deprecated To determine the maximum Rx gain, use skiq_read_rx_gain_index_range().

Definition at line 517 of file sidekiq_api.h.

6.1.2.15 #define SKIQ_MAX_NUM_CARDS (32)

SKIQ_MAX_NUM_CARDS is the maximum number of Sidekiq cards that is supported in a system

Definition at line 521 of file sidekiq_api.h.

6.1.2.16 #define SKIQ_SYS_TIMESTAMP_FREQ (40000000)

SKIQ_SYS_TIMESTAMP_FREQ is the frequency at which the system timestamp increments

Attention

This value is valid only for mPCIe and M.2

Deprecated All platforms should use the skiq_read_sys_timestamp_freq() API instead

Definition at line 530 of file sidekiq_api.h.

6.1.2.17 #define SKIQ_RX_SYS_META_WORD_OFFSET (4)

SKIQ_RX_SYS_META_WORD_OFFSET is the offset at which the system metadata is located within a receive packet. Included in this is the Rx handle as well as the overload bit

Deprecated Use skiq_rx_block_t::hdl, skiq_rx_block_t::overload, and skiq_rx_block_t::rfic_control instead of this definition

Definition at line 538 of file sidekiq_api.h.

6.1.2.18 #define SKIQ_RX_USER_META_WORD_OFFSET (5)

SKIQ_RX_USER_META_WORD_OFFSET is the offset at which the user-defined metadata is located with a receive packet

Deprecated Use skiq_rx_block_t::user_meta instead of this definition

Definition at line 544 of file sidekiq_api.h.

6.1.2.19 #define SKIQ_RX_META_HDL_BITS (0x3F)

SKIQ_RX_META_HDL_BITS is the bitmask which represent the Rx handle

Deprecated Use skiq_rx_block_t::hdl instead of this definition

Definition at line 549 of file sidekiq_api.h.

6.1.2.20 #define SKIQ_RX_META_OVERLOAD_BIT (1 << 6)

SKIQ_RX_META_OVERLOAD_BIT is the location of the bit representing the Rx overload detection in the miscellaneous metadata of a receive packet

Deprecated Use skiq_rx_block_t::overload instead of this definition

Definition at line 555 of file sidekiq_api.h.

6.1.2.21 #define SKIQ_RX_META_RFIC_CTRL_BITS (0xFF)

SKIQ_RX_META_RFIC_CTRL_BITS are the bits which contain the RFIC control bits embedded within the system metadata

Deprecated Use skiq_rx_block_t::rfic_control instead of this definition

Definition at line 561 of file sidekiq_api.h.

6.1.2.22 #define SKIQ_RX_META_RFIC_CTRL_OFFSET (7)

SKIQ_RX_META_RFIC_CTRL_OFFSET is the bit offset where the RFIC control bits are located within the system metadata

Deprecated Use skiq_rx_block_t::rfic_control instead of this definition

Definition at line 567 of file sidekiq_api.h.

6.1.2.23 #define RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA1 (0x16)

RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA1 is the value that should be used to enable the gain values for RxA1 to be presented in the system metadata of each receive packet. Use this definition in conjunction with skiq_write_rfic_control_output_config()

Deprecated since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration to present A1 gain in the metadata.

Definition at line 597 of file sidekiq_api.h.

6.1.2.24 #define RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA2 (0x17)

RFIC_CONTROL_OUTPUT_MODE_GAIN_CONTROL_RXA2 is the value that should be used to enable the gain values for RxA2 to be presented in the system metadata of each receive packet. Use this definition in conjunction with skiq_write_rfic_control_output_config()

Deprecated since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration to present A2 gain in the metadata.

Definition at line 608 of file sidekiq_api.h.

6.1.2.25 #define RFIC_CONTROL_OUTPUT_MODE_GAIN_BITS (0x7F)

RFIC_CONTROL_OUTPUT_MODE_GAIN_BITS are the bits used in conveying the current gain setting (read from the RFIC control output). Use this definition in conjunction with skiq_write_rfic_control_output_config()

Deprecated since v4.9.0 Not all radio types use this control output mode value to present the gain in the control output field. Use skiq_read_rfic_control_output_rx_gain_config() to determine appropriate enable and mode configuration.

Definition at line 618 of file sidekiq_api.h.

6.1.3 Function Documentation

6.1.3.1 EPIQ_API int32_t skiq_read_fpga_version (uint8_t *card*, uint32_t ∗ *p_git_hash*, uint32_t ∗ *p_build_date*, uint8_t ∗ *p_major*, uint8_t ∗ *p_minor*, skiq_fpga_tx_fifo_size_t ∗ *p_tx_fifo_size*)

The skiq_read_fpga_version() function is responsible for returning the major/minor revision numbers for the currently loaded FPGA bitstream.

Deprecated Use skiq_read_fpga_semantic_version() and skiq_read_fpga_tx_fifo_size() instead of skiq_read_fpga_version()

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_git_hash*	a pointer to where the 32-bit git hash will be written
out	*p_build_date*	a pointer to where the 32-bit build date will be written
out	*p_major*	a pointer to where the major rev # should be written
out	*p_minor*	a pointer to where the minor rev # should be written
out	*p_tx_fifo_size*	a pointer to where the FPGA's TX FIFO size enumeration should be written

Returns

int32_t status where 0=success, anything else is an error

6.1.3.2 EPIQ_API int32_t skiq_read_hw_version (uint8_t *card*, skiq_hw_vers_t ∗ *p_hw_version*)

The skiq_read_hw_version() function is responsible for returning the hardware version number of the Sidekiq board.

Deprecated

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_hw_version*	[skiq_hw_vers_t] a pointer to hold the hardware version

Returns

int32_t: status where 0=success, anything else is an error

6.1.3.3 EPIQ_API int32_t skiq_read_product_version (uint8_t *card*, skiq_product_t ∗ *p_product*)

The skiq_read_product_version() function is responsible for returning the product version of the Sidekiq board.

Deprecated

Parameters

in	*card*	card index of the Sidekiq of interest
out	*p_product*	[skiq_product_t] a pointer to hold the product version

Returns

int32_t: status where 0=success, anything else is an error

6.1.3.4 EPIQ_API const char∗ skiq_hardware_vers_string (skiq_hw_vers_t *hardware_vers*)

The skiq_hardware_vers_string() function returns a string representation of the passed in hardware version.

Deprecated

Parameters

in	*hardware_vers*	hardware version value

Returns

const char∗ string representing the hardware version

6.1.3.5 EPIQ_API const char∗ skiq_product_vers_string (skiq_product_t *product_vers*)

The skiq_product_vers_string() function returns a string representation of the passed in product version.

Deprecated

Parameters

in	*product_vers*	product version value

Returns

const char∗ string representing the product version

6.1.3.6 EPIQ_API int32_t skiq_read_max_sample_rate (uint8_t *card,* uint32_t ∗ *p_max_sample_rate*)

The skiq_read_max_sample_rate() function returns the maximum sample rate possible for the Sidekiq card requested based on the current channel mode and product.

Since

Function added in API **v4.2.0**

Deprecated This function has been deprecated and may not return the correct maximum sample rate for all handles, this has been replaced with skiq_read_parameters.

Parameters

in	*card*	card index of Sidekiq of interest
out	*p_max_sample_rate*	pointer to where to store the maximum sample rate

Returns

const char∗ string representing the product version

6.1.3.7 EPIQ_API int32_t skiq_read_min_sample_rate (uint8_t *card,* uint32_t ∗ *p_min_sample_rate*)

The skiq_read_min_sample_rate() function returns the minimum sample rate possible for the Sidekiq card requested based on the product.

Since

Function added in API **v4.2.0**

Deprecated This function has been deprecated and may not return the correct minimum sample rate for all handles, this has been replaced with skiq_read_parameters.

Parameters

in	*card*	card index of Sidekiq of interest

out	*p_min_sample_rate*	pointer to where to store the minimum sample rate

Returns

const char* string representing the product version

6.1.3.8 EPIQ_API int32_t skiq_write_rx_sample_shift (uint8_t *card,* skiq_rx_hdl_t *hdl,* uint8_t *shift_delay*)

The skiq_write_rx_sample_shift() function allows the user to set a sample delay value on either A1 or A2. This is currently only supported on the NV100 for SKIQ_MAX_SAMPLE_SHIFT_NV100 samples per channel.

Since

Function added in v4.18.0

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	handle to apply the sample delay shift on
in	*shift_delay*	# samples to delay, valid for [0, 4] range

Returns

0 on success, else a negative errno value

Return values

-EINVAL	Requested shift or handle value is not supported
-ENOTSUP	Sample shift register not supported on this device
-EIO	A fault occurred communicating with the FPGA

6.1.3.9 EPIQ_API int32_t skiq_read_rx_sample_shift (uint8_t *card,* skiq_rx_hdl_t *hdl,* uint8_t * *shift_delay*)

The skiq_read_rx_sample_shift() function allows the user to read a sample delay value on either A1 or A2. This is currently only supported on the NV100

Since

Function added in v4.18.0

Parameters

in	*card*	card index of the Sidekiq of interest
in	*hdl*	handle to read the sample delay shift on

out	*shift_delay*	# samples currently delayed

Returns

0 on success, else a negative errno value

Return values

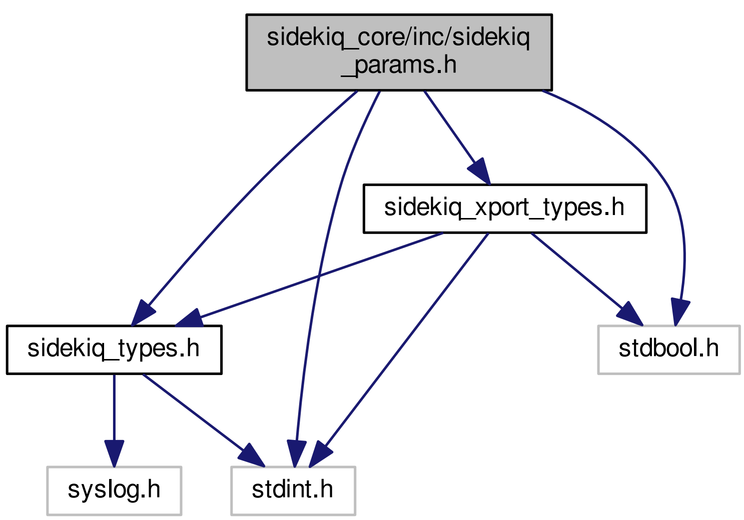
-EINVAL	Requested handle value is not supported
-ENOTSUP	Sample shift register not supported on this device
-EIO	A fault occurred communicating with the FPGA

6.2 sidekiq_core/inc/sidekiq_params.h File Reference

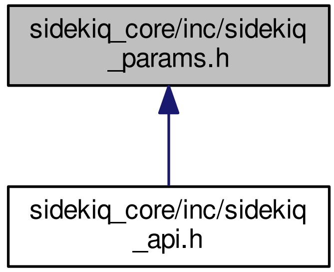
This file contains data structures used to determine the parameters of a given Sidekiq.

```
#include <stdint.h>
#include <stdbool.h>
#include "sidekiq_types.h"
#include "sidekiq_xport_types.h"
```

Include dependency graph for sidekiq_params.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct skiq_card_param_t

  Parameters related to a physical Sidekiq card.
- struct skiq_fpga_param_t

  Parameter for the Sidekiq's on board FPGA.
- struct skiq_fw_param_t

  Parameters for the firmware loaded onto a Sidekiq.
- struct skiq_rf_param_t

  Parameters for the Sidekiq's RF capabilities.
- struct skiq_rx_param_t

  Parameters for each Rx channel on a Sidekiq card.
- struct skiq_tx_param_t

  Parameters for each Tx channel on a Sidekiq card.
- struct skiq_param_t

  Parameters for the entire Sidekiq.

Enumerations

- enum fpga_state_t { FPGA_STATE_VERSION_INACCESSIBLE = 0, FPGA_STATE_VERSION_VALID, FPGA_STATE_VERSION_GOLDEN }

  Field for the state of the running bitstream.

6.2.1 Detailed Description

This file contains data structures used to determine the parameters of a given Sidekiq.

```
Copyright 2017-2021 Epiq Solutions, All Rights Reserved
```

Definition in file sidekiq_params.h.

6.2.2 Enumeration Type Documentation

6.2.2.1 enum fpga_state_t

Field for the state of the running bitstream.

Enumerator

FPGA_STATE_VERSION_INACCESSIBLE

FPGA_STATE_VERSION_VALID

FPGA_STATE_VERSION_GOLDEN

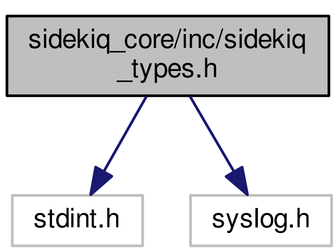
Definition at line 56 of file sidekiq_params.h.

6.3 sidekiq_core/inc/sidekiq_types.h File Reference

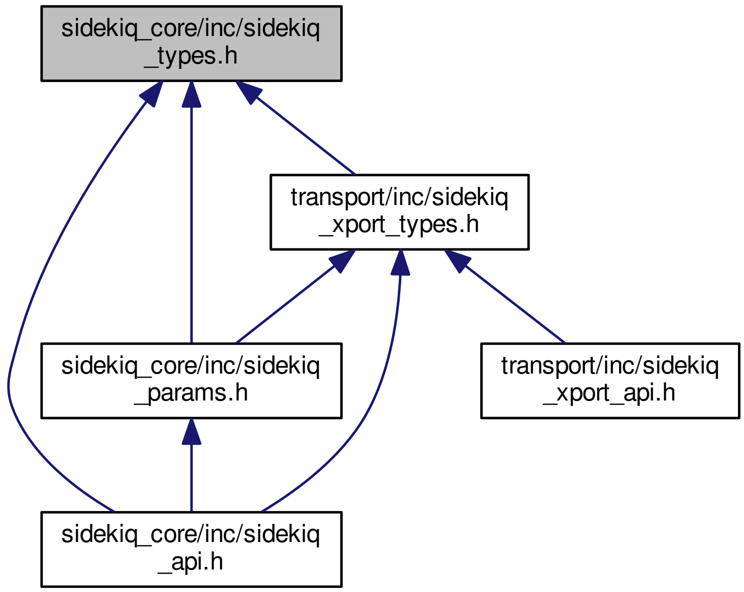
This file contains the public type definitions of the sidekiq_api provided by libsidekiq.

```
#include <stdint.h>
#include <syslog.h>
```

Include dependency graph for sidekiq_types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct skiq_part_info_t

  Sidekiq Part Information.
- struct skiq_tx_block_t

  Sidekiq Transmit Block type definition for use with skiq_transmit and skiq_tx_callback_t.
- struct skiq_rx_block_t

  Sidekiq Receive Block type definition for use with skiq_receive.

Macros

- #define EPIQ_API extern
- #define SKIQ_MAX_SAMPLE_SHIFT_NV100 4

  SKIQ_MAX_SAMPLE_SHIFT_NV100 defines the maximum sample shift value used by skiq_write_rx_sample_shift() function. This is currently supported only for NV100
- #define SKIQ_SERIAL_NUM_STRLEN (6)

  Number of bytes contained in the serial number (including '\0')
- #define SKIQ_PART_NUM_STRLEN (7)

  Number of bytes contained in the part number (including '\0')
- #define SKIQ_REVISION_STRLEN (3)

  Number of bytes contained in the revision (including '\0')
- #define SKIQ_VARIANT_STRLEN (3)

Number of bytes contained in the variant (including '\0')
- #define SKIQ_MAX_NUM_FILTERS (20)

    Maximum number of filters available for a handle.
- #define SKIQ_MAX_NUM_TX_QUEUED_PACKETS (50)

    Maximum number of TX packets that can be queued when running in skiq_tx_transfer_mode_async.
- #define SKIQ_MAX_NUM_FREQ_HOPS (512)

    Maximum number of frequencies that can be specified in a hopping list.
- #define SKIQ_TX_BLOCK_MEMORY_ALIGN 4096

    Defines the memory alignment of a transmit block when allocated using SKIQ_TX_BLOCK_INITIALIZER, SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS, SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES, skiq_tx_block_allocate() or skiq_tx_block_allocate_by_bytes()
- #define SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES(var_name, data_size_in_bytes)
- #define SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS(var_name, data_size_in_words)
- #define SKIQ_TX_BLOCK_INITIALIZER(var_name)
- #define SKIQ_RX_BLOCK_INITIALIZER_BY_BYTES(var_name, data_size_in_bytes)
- #define SKIQ_RX_BLOCK_INITIALIZER_BY_WORDS(var_name, data_size_in_words)
- #define SKIQ_RX_BLOCK_INITIALIZER(var_name)
- #define SKIQ_LOG_DEBUG ((int)LOG_DEBUG)
- #define SKIQ_LOG_INFO ((int)LOG_INFO)
- #define SKIQ_LOG_WARNING ((int)LOG_WARNING)
- #define SKIQ_LOG_ERROR ((int)LOG_ERR)

Typedefs

- typedef void(∗ skiq_tx_callback_t)(int32_t status, skiq_tx_block_t ∗p_block, void ∗p_user)

    Transmit callback function type definition.
- typedef void(∗ skiq_tx_ena_callback_t)(uint8_t card, int32_t status)

    Transmit enabled callback function type definition where card is the Sidekiq card whose transmitter has been enabled and status is the error code associated with enabling the transmitter (0 is success)
- typedef float _Complex float_complex_t

Enumerations

- enum skiq_tx_timestamp_base_t { skiq_tx_rf_timestamp =0, skiq_tx_system_timestamp }

    Tx supports transmitting on System Timestamp or RF Timestamp on certain Sidekiq products.
- enum skiq_tx_flow_mode_t { skiq_tx_immediate_data_flow_mode =0, skiq_tx_with_timestamps_data_flow_mode, skiq_tx_with_timestamps_allow_late_data_flow_mode }

    There are several different data flow modes that can be used when transmitting data on a Sidekiq Tx interface:
- enum skiq_tx_transfer_mode_t { skiq_tx_transfer_mode_sync =0, skiq_tx_transfer_mode_async }

    There are different transfer modes that can be used when transmitting data:
- enum skiq_data_src_t { skiq_data_src_iq =0, skiq_data_src_counter }

    An Rx interface is typically configured to generate complex I/Q samples. However, there are test cases where it is useful to have the I/Q data replaced with an incrementing counter. This is treated as a different "data source", which can be configured by the user at run-time before an Rx interface is started.
- enum skiq_iq_order_t { skiq_iq_order_qi =0, skiq_iq_order_iq }

    An interface is configured to transmit or receive complex I/Q samples. By default samples are received/transmitted as I/Q pairs with 'Q' sample occurring first, followed by the 'I' sample. Ordering can be configured by the user at run-time before an Rx interface is started.

- enum skiq_rx_stream_mode_t { skiq_rx_stream_mode_high_tput =0, skiq_rx_stream_mode_low_latency, skiq_rx_stream_mode_balanced, skiq_rx_stream_mode_end }

  Sidekiq supports three different receive stream modes that change the relative IQ sample block latency (skiq_rx_block_t) between the FPGA and host CPU. The skiq_rx_stream_mode_high_tput setting is business as usual and provides the same receive latency that exists in the previous releases of libsidekiq. The skiq_rx_stream_mode_low_latency setting provides a smaller block of IQ samples from skiq_receive() more often and effectively lowers the latency from RF reception to host CPU. The skiq_rx_stream_mode_balanced is a compromise between the high_tput and low_latency modes which has a reduced overall throughput relative to high_tput but results in a larger number of samples per packet than the low_latency mode.

- enum skiq_trigger_src_t { skiq_trigger_src_immediate =0, skiq_trigger_src_1pps, skiq_trigger_src_synced }

  A trigger source may be specified when starting or stopping multiple handle streaming. The trigger may be specified as 'immediate' and happens without any synchronization between handles. It may also be specified as '1PPS' so all specified handles would start streaming synchronized on a PPS edge. If the FPGA bitstream supports it (>= 3.11.0), a value of skiq_trigger_src_synced causes all specified handles to start or stop streaming immediately, but also synchronized (RF timestamps are aligned). A similar application may be used when stopping multiple handles from streaming.

- enum skiq_rx_hdl_t {
  skiq_rx_hdl_A1 =0, skiq_rx_hdl_A2 =1, skiq_rx_hdl_B1 =2, skiq_rx_hdl_B2 =3,
  skiq_rx_hdl_C1 =4, skiq_rx_hdl_D1 =5, skiq_rx_hdl_end }

  Sidekiq supports several Rx interface handles. The skiq_rx_hdl_t enum is used to define the different Rx interface handles.

- enum skiq_tx_hdl_t {
  skiq_tx_hdl_A1 =0, skiq_tx_hdl_A2 =1, skiq_tx_hdl_B1 =2, skiq_tx_hdl_B2 =3,
  skiq_tx_hdl_end }

  Sidekiq supports a single Tx interface handles. The skiq_tx_hdl_t enum is used to define the Tx interface handle.

- enum skiq_filt_t {
  skiq_filt_invalid = -1, skiq_filt_0_to_3000_MHz =0, skiq_filt_3000_to_6000_MHz, skiq_filt_0_to_440MHz,
  skiq_filt_440_to_6000MHz, skiq_filt_440_to_580MHz, skiq_filt_580_to_810MHz, skiq_filt_810_to_1170MHz,
  skiq_filt_1170_to_1695MHz, skiq_filt_1695_to_2540MHz, skiq_filt_2540_to_3840MHz, skiq_filt_3840_to_6000MHz,
  skiq_filt_0_to_300MHz, skiq_filt_300_to_6000MHz, skiq_filt_50_to_435MHz, skiq_filt_435_to_910MHz,
  skiq_filt_910_to_1950MHz, skiq_filt_1950_to_6000MHz, skiq_filt_0_to_6000MHz, skiq_filt_390_to_620MHz,
  skiq_filt_540_to_850MHz, skiq_filt_770_to_1210MHz, skiq_filt_1130_to_1760MHz, skiq_filt_1680_to_2580MHz,
  skiq_filt_2500_to_3880MHz, skiq_filt_3800_to_6000MHz, skiq_filt_47_to_135MHz, skiq_filt_135_to_145MHz,
  skiq_filt_145_to_150MHz, skiq_filt_150_to_162MHz, skiq_filt_162_to_175MHz, skiq_filt_175_to_190MHz,
  skiq_filt_190_to_212MHz, skiq_filt_212_to_230MHz, skiq_filt_230_to_280MHz, skiq_filt_280_to_366MHz,
  skiq_filt_366_to_475MHz, skiq_filt_475_to_625MHz, skiq_filt_625_to_800MHz, skiq_filt_800_to_1175MHz,
  skiq_filt_1175_to_1500MHz, skiq_filt_1500_to_2100MHz, skiq_filt_2100_to_2775MHz, skiq_filt_2775_to_3360MHz,
  skiq_filt_3360_to_4600MHz, skiq_filt_4600_to_6000MHz, skiq_filt_30_to_450MHz, skiq_filt_450_to_600MHz,
  skiq_filt_600_to_800MHz, skiq_filt_800_to_1200MHz, skiq_filt_1200_to_1700MHz, skiq_filt_1700_to_2700MHz,
  skiq_filt_2700_to_3600MHz, skiq_filt_3600_to_6000MHz, skiq_filt_max }

Each RF path in Sidekiq has integrated filter options that can be software-controlled. By default, the filter is automatically selected based on the requested LO frequency. The skiq_filt_t enum is used to specify a filter selection. Note: not all filter options are available for hardware variants. Available filter variants can be queried with skiq_read_rx_filters_avail.

- enum skiq_rx_gain_t { skiq_rx_gain_manual = 0, skiq_rx_gain_auto }

  Rx gain can be controlled either manually or automatically. The skiq_rx_gain_t enum is used to specify the mode of gain control.
- enum skiq_rx_attenuation_mode_t { skiq_rx_attenuation_mode_manual =0, skiq_rx_attenuation_mode_noise_figure, skiq_rx_attenuation_mode_normalized }

  Rx attenuation mode.
- enum skiq_chan_mode_t { skiq_chan_mode_single =0, skiq_chan_mode_dual }

  Sidekiq can run either in single Rx or dual channel Rx mode. The skiq_chan_mode_t enum is used to specify the Rx/Tx channel mode.
- enum skiq_part_t {
  skiq_mpcie =0, skiq_m2, skiq_x2, skiq_z2,
  skiq_x4, skiq_m2_2280, skiq_z2p, skiq_z3u,
  skiq_nv100, skiq_part_invalid }

  Sidekiq Part.
- enum skiq_hw_vers_t {
  skiq_hw_vers_mpcie_a =1, skiq_hw_vers_mpcie_b =2, skiq_hw_vers_mpcie_c =3, skiq_hw_vers_mpcie_d =4,
  skiq_hw_vers_mpcie_e =5, skiq_hw_vers_m2_b =6, skiq_hw_vers_m2_c =7, skiq_hw_vers_m2_d =8,
  skiq_hw_vers_reserved, skiq_hw_vers_mpcie_masquerade = skiq_hw_vers_mpcie_c, skiq_hw_vers_m2_masquerade = skiq_hw_vers_m2_c, skiq_hw_vers_invalid =0xFFF }

  Sidekiq has multiple revisions of the hardware. The skiq_hw_vers_t enum defines the revisions that have been deployed.
- enum skiq_product_t {
  skiq_product_mpcie_001 =0, skiq_product_mpcie_002 =1, skiq_product_m2_001 =2, skiq_product_m2_002 =3,
  skiq_product_reserved, skiq_product_invalid =0xF }

  There are multiple products controllable through libsidekiq. The skiq_product_t enum defines the Sidekiq product types.
- enum skiq_rx_fir_gain_t { skiq_rx_fir_gain_neg_12 =3, skiq_rx_fir_gain_neg_6 =2, skiq_rx_fir_gain_0 =1, skiq_rx_fir_gain_6 =0 }

  Rx FIR filter gain settings, applied to the Rx FIR used in the Rx channel bandwidth configuration.
- enum skiq_tx_fir_gain_t { skiq_tx_fir_gain_neg_6 =1, skiq_tx_fir_gain_0 =0 }

  Tx FIR filter gain settings, applied to the Tx FIR used in the Tx channel bandwidth configuration.
- enum skiq_ref_clock_select_t {
  skiq_ref_clock_internal =0, skiq_ref_clock_external, skiq_ref_clock_host, skiq_ref_clock_carrier_edge,
  skiq_ref_clock_invalid }

  Reference clock setting.
- enum skiq_fpga_tx_fifo_size_t {
  skiq_fpga_tx_fifo_size_unknown = 0, skiq_fpga_tx_fifo_size_4k = 1, skiq_fpga_tx_fifo_size_8k = 2,
  skiq_fpga_tx_fifo_size_16k = 3,
  skiq_fpga_tx_fifo_size_32k = 4, skiq_fpga_tx_fifo_size_64k = 5 }

  FPGA Tx FIFO Size. The FIFO size is the number of packets the FPGA can hold prior to actually transmitting the data.
- enum skiq_rx_status_t {
  skiq_rx_status_success = 0, skiq_rx_status_no_data = -1, skiq_rx_status_error_generic = -6, skiq_rx_status_error_overrun = -11,
  skiq_rx_status_error_packet_malformed = -12, skiq_rx_status_error_card_not_active = -19, skiq_rx_status_error_not_streaming = -29 }

Possible return codes from skiq_receive.

- enum skiq_rf_port_config_t { skiq_rf_port_config_fixed = 0, skiq_rf_port_config_tdd, skiq_rf_port_config_trx, skiq_rf_port_config_invalid }

  RF port configuration options of Sidekiq.
- enum skiq_rf_port_t {
  skiq_rf_port_unknown =-1, skiq_rf_port_J1 = 0, skiq_rf_port_J2, skiq_rf_port_J3,
  skiq_rf_port_J4, skiq_rf_port_J5, skiq_rf_port_J6, skiq_rf_port_J7,
  skiq_rf_port_J300, skiq_rf_port_Jxxx_RX1, skiq_rf_port_Jxxx_TX1RX2, skiq_rf_port_J8,
  skiq_rf_port_max }

  RF ports of Sidekiq.
- enum skiq_tx_quadcal_mode_t { skiq_tx_quadcal_mode_auto =0, skiq_tx_quadcal_mode_manual }

  TX Quadrature Calibration Mode.
- enum skiq_rx_cal_mode_t { skiq_rx_cal_mode_auto =0, skiq_rx_cal_mode_manual }

  RX Calibration Mode.
- enum skiq_rx_cal_type_t { skiq_rx_cal_type_none = 0x00000000, skiq_rx_cal_type_dc_offset = 0x00000001, skiq_rx_cal_type_quadrature = 0x00000002 }

  RX Calibration Types.
- enum skiq_1pps_source_t { skiq_1pps_source_unavailable =-1, skiq_1pps_source_external =0, skiq_1pps_source_host =1 }

  Source of 1PPS. Note that not all products support all configurations.
- enum skiq_freq_tune_mode_t { skiq_freq_tune_mode_standard =0, skiq_freq_tune_mode_hop_immediate, skiq_freq_tune_mode_hop_on_timestamp }

  Frequency Tune mode. Note that not all products support all configurations.
- enum skiq_fmc_carrier_t {
  skiq_fmc_carrier_not_applicable, skiq_fmc_carrier_unknown, skiq_fmc_carrier_ams_wb3xzd, skiq_fmc_carrier_htg_k800,
  skiq_fmc_carrier_ams_wb3xbm, skiq_fmc_carrier_htg_k810 }
- enum skiq_fpga_device_t {
  skiq_fpga_device_unknown, skiq_fpga_device_xc6slx45t, skiq_fpga_device_xc7a50t, skiq_fpga_device_xc7z010,
  skiq_fpga_device_xcku060, skiq_fpga_device_xcku115, skiq_fpga_device_xczu3eg }
- enum skiq_rfic_pin_mode_t { skiq_rfic_pin_control_sw =0, skiq_rfic_pin_control_fpga_gpio }
- enum skiq_gpsdo_support_t {
  skiq_gpsdo_support_unknown = 0, skiq_gpsdo_support_is_supported, skiq_gpsdo_support_card_not_supported, skiq_gpsdo_support_fpga_not_supported,
  skiq_gpsdo_support_not_supported }

  The status of GPSDO support on a given card / FPGA bitstream.

Variables

- EPIQ_API const char ∗ SKIQ_FILT_STRINGS [skiq_filt_max]

  String representation of skiq_filt_t enumeration.
- EPIQ_API const char ∗ SKIQ_RX_STREAM_MODE_STRINGS [skiq_rx_stream_mode_end]

  String representation of skiq_rx_stream_mode_t.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_MPCIE_001

  String representation of the Sidekiq mPCIe 001 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_MPCIE_002

  String representation of the Sidekiq mPCIe 002 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_M2

String representation of the Sidekiq m.2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_X2

  String representation of the Sidekiq X2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z2

  String representation of the Sidekiq Z2 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_X4

  String representation of the Sidekiq X4 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_M2_2280

  String representation of the Sidekiq M.2 2280 part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z2P

  String representation of the Sidekiq Z2+ part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_Z3U

  String representation of the Sidekiq Z3u part.
- EPIQ_API const char ∗ SKIQ_PART_NUM_STRING_NV100

  String representation of the Sidekiq NV100 part.
- EPIQ_API const char ∗ SKIQ_RF_PORT_STRINGS [skiq_rf_port_max]

  String representation of skiq_rf_port_t enumeration.

6.3.1 Detailed Description

This file contains the public type definitions of the sidekiq_api provided by libsidekiq.

Definition in file sidekiq_types.h.

6.3.2 Macro Definition Documentation

6.3.2.1 #define EPIQ_API extern

Definition at line 48 of file sidekiq_types.h.

6.3.2.2 #define SKIQ_MAX_SAMPLE_SHIFT_NV100 4

SKIQ_MAX_SAMPLE_SHIFT_NV100 defines the maximum sample shift value used by skiq_write_rx_sample_shift() function. This is currently supported only for NV100

Definition at line 70 of file sidekiq_types.h.

6.3.2.3 #define SKIQ_LOG_DEBUG ((int)LOG_DEBUG)

Sidekiq logging levels. The lower the numeric value, the higher the severity level.

Definition at line 1269 of file sidekiq_types.h.

6.3.2.4 #define SKIQ_LOG_INFO ((int)LOG_INFO)

Definition at line 1270 of file sidekiq_types.h.

6.3.2.5 #define SKIQ_LOG_WARNING ((int)LOG_WARNING)

Definition at line 1271 of file sidekiq_types.h.

6.3.2.6 #define SKIQ_LOG_ERROR ((int)LOG_ERR)

Definition at line 1272 of file sidekiq_types.h.

6.3.3 Typedef Documentation

6.3.3.1 typedef float _Complex float_complex_t

A 'float _Complex' type definition that provides a cross-platform complex variable type. Under Linux and MinGW64, the float_complex_t definition resolves to 'float complex' while under Visual Studio, it resolves to '_Fcomplex'

Definition at line 1261 of file sidekiq_types.h.

6.3.4 Enumeration Type Documentation

6.3.4.1 enum skiq_hw_vers_t

Sidekiq has multiple revisions of the hardware. The skiq_hw_vers_t enum defines the revisions that have been deployed.

Note

These constants are NOT used in the EEPROM; hardware.c maps skiq_eeprom_hw_vers_ to this constant. You can safely change these constants without worrying about backwards compatability (in terms of EE).

Deprecated

Enumerator

skiq_hw_vers_mpcie_a

skiq_hw_vers_mpcie_b

skiq_hw_vers_mpcie_c

skiq_hw_vers_mpcie_d

skiq_hw_vers_mpcie_e

skiq_hw_vers_m2_b

skiq_hw_vers_m2_c

skiq_hw_vers_m2_d

skiq_hw_vers_reserved

skiq_hw_vers_mpcie_masquerade

skiq_hw_vers_m2_masquerade

skiq_hw_vers_invalid

Definition at line 622 of file sidekiq_types.h.

6.3.4.2 enum skiq_product_t

There are multiple products controllable through libsidekiq. The skiq_product_t enum defines the Sidekiq product types.

Deprecated

Enumerator

skiq_product_mpcie_001 supports RxA1/A2 and TxA1

skiq_product_mpcie_002 supports RxA1 and TxA1 only

skiq_product_m2_001 supports RxA1/A2 and TxA1/A2

skiq_product_m2_002 supports RxA1 and TxA1

skiq_product_reserved

skiq_product_invalid

Definition at line 654 of file sidekiq_types.h.

6.3.4.3 enum skiq_fmc_carrier_t

Enumerator

skiq_fmc_carrier_not_applicable Applies to those Sidekiq devices that are not FMC form factor.

skiq_fmc_carrier_unknown Queries of the Sidekiq device failed to find a supported FMC carrier.

skiq_fmc_carrier_ams_wb3xzd Annapolis Micro Systems WILDSTAR WB3XZD (`https://www.-annapmicro.com/products/wildstar-ultrakvp-zp-dram-3u-openvpx/`)

skiq_fmc_carrier_htg_k800 HiTech Global K800 (`http://www.hitechglobal.com/Boards/Kintex--UltraScale.htm`)

skiq_fmc_carrier_ams_wb3xbm Annapolis Micro Systems WILDSTAR WB3XBM (`https://www.-annapmicro.com/products/wildstar-3xbm-3u-openvpx-fpga-processor-wb3xbm/`)

skiq_fmc_carrier_htg_k810 HiTech Global K810 (`http://www.hitechglobal.com/Boards/Kintex-UltraScale_COMExpress.htm`)

Definition at line 903 of file sidekiq_types.h.

6.3.4.4 enum skiq_fpga_device_t

Enumerator

skiq_fpga_device_unknown Queries of the Sidekiq device failed to find a supported FPGA device.

skiq_fpga_device_xc6slx45t Xilinx Spartan-6 LXT.

skiq_fpga_device_xc7a50t Xilinx Artix-7 50T.

skiq_fpga_device_xc7z010 Xilinx Zynq-7000.

skiq_fpga_device_xcku060 Xilinx Kintex Ultrascale 60.

skiq_fpga_device_xcku115 Xilinx Kintex Ultrascale 115.

skiq_fpga_device_xczu3eg Xilinx Zynq Ultrascale+ ZU3.

Definition at line 935 of file sidekiq_types.h.

6.3.4.5 enum skiq_rfic_pin_mode_t

Enumerator

skiq_rfic_pin_control_sw

skiq_rfic_pin_control_fpga_gpio

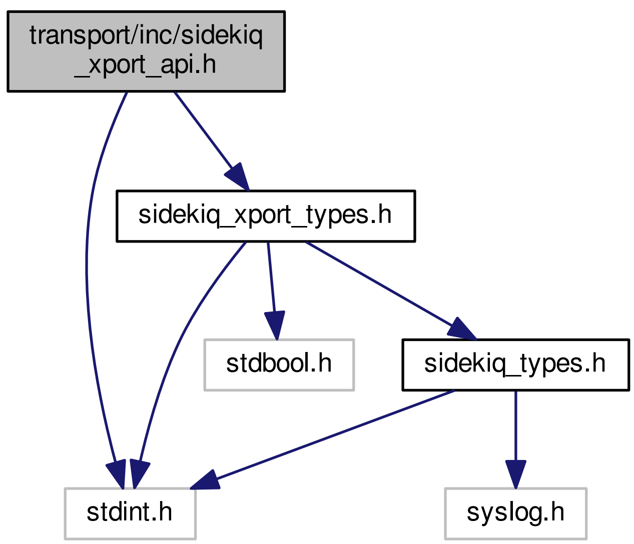
Definition at line 960 of file sidekiq_types.h.

6.4 transport/inc/sidekiq_xport_api.h File Reference

This file contains the public interface of the sidekiq_xport_api provided by libsidekiq.

```
#include <stdint.h>
#include "sidekiq_xport_types.h"
```

Include dependency graph for sidekiq_xport_api.h:



Functions

- EPIQ_API int32_t skiq_register_custom_transport (skiq_xport_card_functions_t ∗functions)
- EPIQ_API int32_t skiq_unregister_custom_transport (void)
- EPIQ_API int32_t xport_register_fpga_functions (skiq_xport_id_t ∗p_xport_id, skiq_xport_fpga_functions_t ∗functions)
- EPIQ_API int32_t xport_register_rx_functions (skiq_xport_id_t ∗p_xport_id, skiq_xport_rx_functions_t ∗functions)
- EPIQ_API int32_t xport_register_tx_functions (skiq_xport_id_t ∗p_xport_id, skiq_xport_tx_functions_t ∗functions)

- EPIQ_API int32_t xport_unregister_fpga_functions (skiq_xport_id_t *p_xport_id)
- EPIQ_API int32_t xport_unregister_rx_functions (skiq_xport_id_t *p_xport_id)
- EPIQ_API int32_t xport_unregister_tx_functions (skiq_xport_id_t *p_xport_id)

6.4.1 Detailed Description

This file contains the public interface of the sidekiq_xport_api provided by libsidekiq.

Definition in file sidekiq_xport_api.h.

6.4.2 Function Documentation

6.4.2.1 EPIQ_API int32_t skiq_register_custom_transport (skiq_xport_card_functions_t * *functions*)

The skiq_register_custom_transport function registers a set of custom transport card functions. At a minimum, the .card_probe and .card_init function pointers must point to valid implementations and cannot be NULL. Only one custom transport implementation may be registered and is accessed indirectly by specifying skiq_xport_type_custom in calls to skiq_init().

Parameters

in	*functions*	reference to a skiq_xport_card_functions_t struct to register

Returns

int32_t status where 0=success, anything else is an error

6.4.2.2 EPIQ_API int32_t skiq_unregister_custom_transport (void)

The skiq_unregister_custom_transport function unregisters (removes) the current custom transport card functions.

Returns

int32_t status where 0=success, anything else is an error

6.4.2.3 EPIQ_API int32_t xport_register_fpga_functions (skiq_xport_id_t * *p_xport_id*, skiq_xport_fpga_functions_t * *functions*)

The xport_register_fpga_functions function is to be used by a custom transport .card_init implementation to register a set of FPGA functions.

Parameters

in	*p_xport_id*	pointer to the transport identifier
in	*functions*	reference to a skiq_xport_fpga_functions_t struct to register

Returns

int32_t status where 0=success, anything else is an error

6.4.2.4 EPIQ_API int32_t xport_register_rx_functions (skiq_xport_id_t ∗ *p_xport_id,* skiq_xport_rx_functions_t ∗ *functions*)

The xport_register_rx_functions function is to be used by a custom transport .card_init implementation to register a set of RX functions. RX functions are used in calls to skiq_start_rx_streaming, skiq_start_rx_-streaming_on_1pps, skiq_stop_rx_streaming, skiq_stop_rx_streaming_on_1pps, skiq_write_rx_sample_rate-_and_bandwidth, and skiq_receive.

Parameters

in	*p_xport_id*	pointer to the transport identifier
in	*functions*	reference to a skiq_xport_rx_functions_t struct to register

Returns

int32_t status where 0=success, anything else is an error

6.4.2.5 EPIQ_API int32_t xport_register_tx_functions (skiq_xport_id_t ∗ *p_xport_id,* skiq_xport_tx_functions_t ∗ *functions*)

The xport_register_tx_functions function is to be used by a custom transport .card_init implementation to register a set of TX functions. TX functions are used in calls to skiq_start_tx_streaming, skiq_start_tx_-streaming_on_1pps, skiq_stop_tx_streaming, skiq_stop_tx_streaming_on_1pps, and skiq_receive.

Parameters

in	*p_xport_id*	pointer to the transport identifier
in	*functions*	reference to a skiq_xport_tx_functions_t struct to register

Returns

int32_t status where 0=success, anything else is an error

6.4.2.6 EPIQ_API int32_t xport_unregister_fpga_functions (skiq_xport_id_t ∗ *p_xport_id*)

The xport_unregister_fpga_functions function is to be used by a custom transport .card_init and/or .card_exit implementation to clear the set of FPGA functions.

Parameters

in	*p_xport_id*	pointer to the transport identifier

Returns

int32_t status where 0=success, anything else is an error

6.4.2.7 EPIQ_API int32_t xport_unregister_rx_functions (skiq_xport_id_t ∗ *p_xport_id*)

The xport_unregister_rx_functions function is to be used by a custom transport .card_init and/or .card_exit implementation to clear the set of RX functions.

Parameters

in	*p_xport_id*	pointer to the transport identifier

Returns

int32_t status where 0=success, anything else is an error

6.4.2.8 EPIQ_API int32_t xport_unregister_tx_functions (skiq_xport_id_t ∗ *p_xport_id*)

The xport_unregister_tx_functions function is to be used by a custom transport .card_init and/or .card_exit implementation to clear the set of TX functions.

Parameters

in	*p_xport_id*	pointer to the transport identifier

Returns

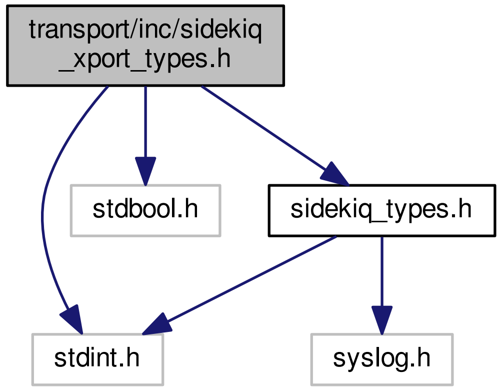
int32_t status where 0=success, anything else is an error

6.5 transport/inc/sidekiq_xport_types.h File Reference

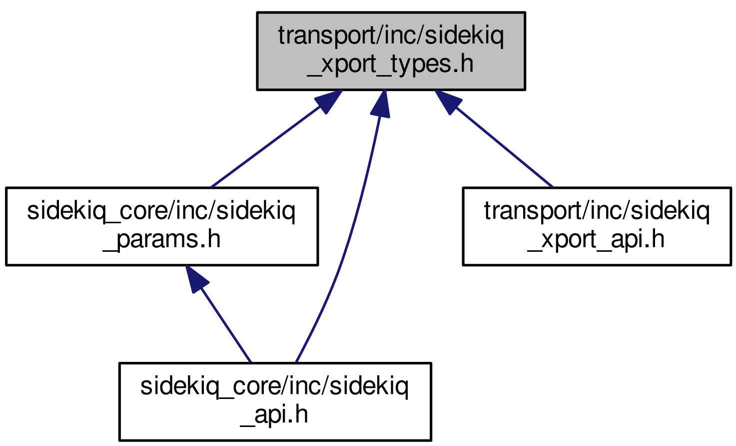
This file contains the type definitions associated with the Sidekiq Transport API (sidekiq_xport_api.h). The skiq_xport_type_t and skiq_xport_init_level_t specify which transport and at which level to perform card initialization. The other function pointer structs are used by custom transport developers to implement a transport layer for use by libsidekiq.

```
#include <stdint.h>
#include <stdbool.h>
#include "sidekiq_types.h"
```

Include dependency graph for sidekiq_xport_types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct skiq_xport_id_t
- struct skiq_xport_card_functions_t
- struct skiq_xport_fpga_functions_t
- struct skiq_xport_rx_functions_t

- struct skiq_xport_tx_functions_t

Macros

- #define SKIQ_XPORT_UID_INVALID (UINT64_MAX)
- #define SKIQ_XPORT_ID_INITIALIZER

    initializer for the skiq_xport_id_t structure

Enumerations

- enum skiq_xport_type_t {
  skiq_xport_type_pcie = 0, skiq_xport_type_usb, skiq_xport_type_custom, skiq_xport_type_net,
  skiq_xport_type_max, skiq_xport_type_auto, skiq_xport_type_unknown }
- enum skiq_xport_init_level_t { skiq_xport_init_level_basic = 0, skiq_xport_init_level_full, skiq_xport_init_level_unknown }

6.5.1 Detailed Description

This file contains the type definitions associated with the Sidekiq Transport API (sidekiq_xport_api.h). The skiq_xport_type_t and skiq_xport_init_level_t specify which transport and at which level to perform card initialization. The other function pointer structs are used by custom transport developers to implement a transport layer for use by libsidekiq.

```
Copyright 2016-2021 Epiq Solutions, All Rights Reserved
```

Definition in file sidekiq_xport_types.h.

6.5.2 Macro Definition Documentation

6.5.2.1 #define SKIQ_XPORT_UID_INVALID (UINT64_MAX)

Definition at line 31 of file sidekiq_xport_types.h.

6.5.2.2 #define SKIQ_XPORT_ID_INITIALIZER

Value:

```
{                                          \
        .xport_uid = SKIQ_XPORT_UID_INVALID,    \
        .type = skiq_xport_type_max,            \
    }
```

initializer for the skiq_xport_id_t structure

Definition at line 34 of file sidekiq_xport_types.h.

6.5.3 Enumeration Type Documentation

6.5.3.1 enum skiq_xport_type_t

The skiq_xport_type_t enumeration is used to specify a transport or combination of transports.

Enumerator

skiq_xport_type_pcie communicate with Sidekiq entirely over PCIe

skiq_xport_type_usb communicate with Sidekiq entirely over USB

skiq_xport_type_custom communicate with Sidekiq entirely using the registered transport implementation provided by a call to skiq_register_custom_transport().

skiq_xport_type_net communicate with Sidekiq entirely over network interface

skiq_xport_type_max INTERNAL USE ONLY

skiq_xport_type_auto automatically detect the transports available and use the preferred one

skiq_xport_type_unknown INTERNAL USE ONLY

Definition at line 47 of file sidekiq_xport_types.h.

6.5.3.2 enum skiq_xport_init_level_t

The skiq_xport_init_level_t enumeration is used to specify an initialization level for a specified transport type. There are two available types.

Enumerator

skiq_xport_init_level_basic minimal initialization necessary to bring up the requested transport interface for register reads/writes, and initialize the mutexes that serializes access to libsidekiq

skiq_xport_init_level_full Same as level_basic + perform the complete bring up of all hardware (most applications concerned with sending/receiving RF will use this)

skiq_xport_init_level_unknown INTERNAL USE ONLY

Definition at line 79 of file sidekiq_xport_types.h.

Chapter 7

Sidekiq Transport Layer

This page is a discussion of the existing Sidekiq transport layers available to developers and includes options for implementing and using a custom transport layer.

7.1 Overview

Sidekiq was developed under the assumption that there would always be either a PCIe or USB interface available to connect the host system and the card itself. This has worked out reasonably well up until now, with a common libsidekiq library capable of supporting either of these interfaces. Most developers will only need to use the "stock" transport layers. The software/FPGA architecture for using PCIe as the transport interface is shown below.

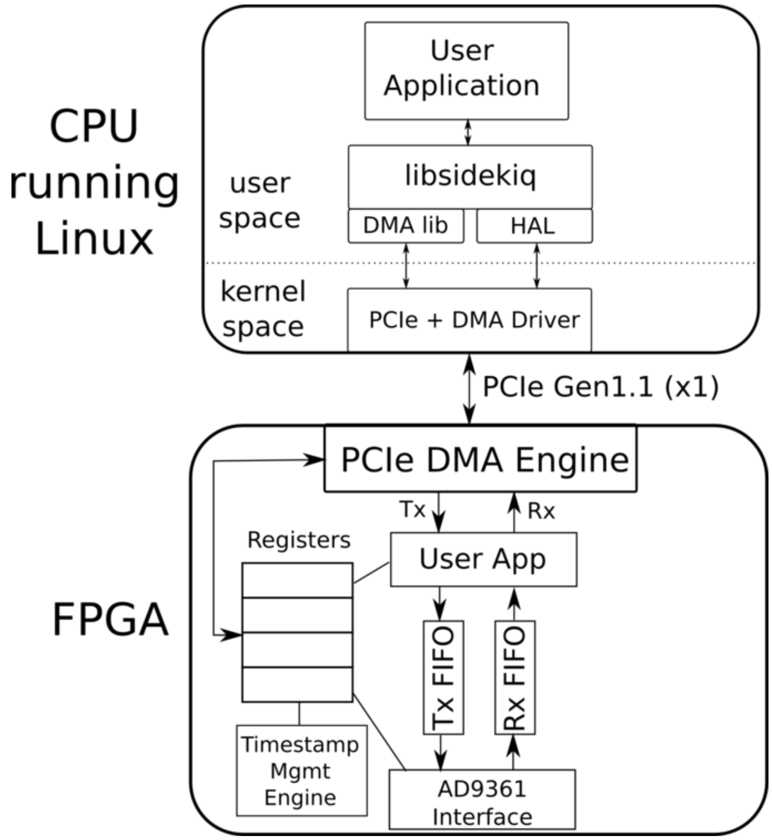


Figure 7.1: Software / FPGA Architecture

However, for some applications, it is necessary to provide a custom transport interface between libsidekiq and the FPGA. In this case, the architecture to support this would need to have a custom software layer at the bottom of the software stack, as well as a custom FPGA interface. Hosting a Sidekiq card in custom hardware may benefit from a custom transport layer. This alternate architecture with a custom transport is shown below.

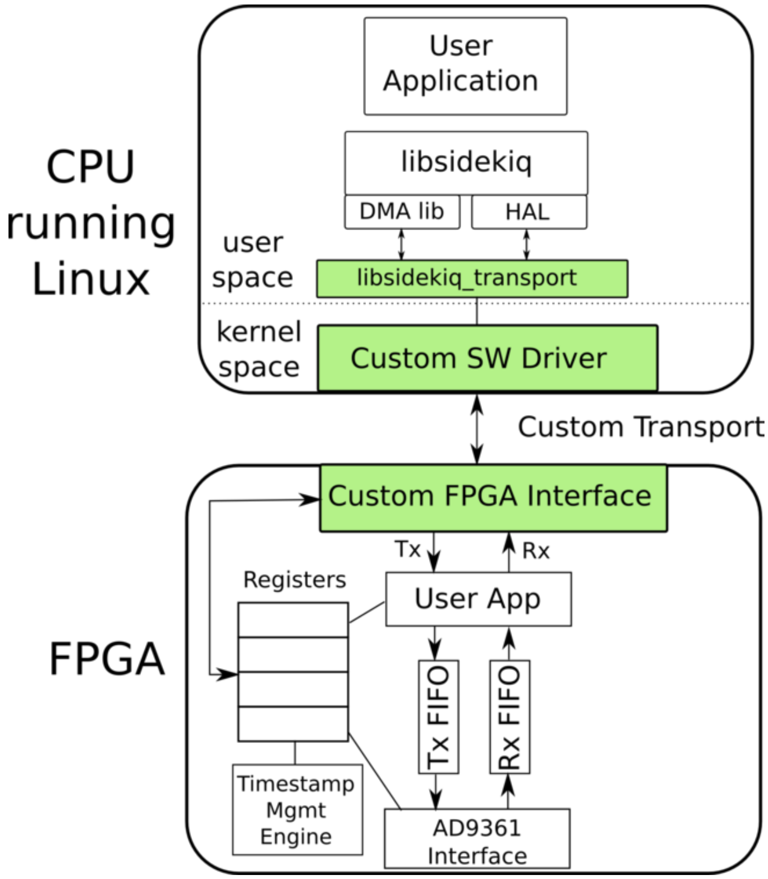


Figure 7.2: Software / FPGA Architecture - Custom Transport

7.2 Custom Transport Interface

Support for a custom transport implementation is available in the latest libsidekiq release starting with v3.2.0 and necessitates the development of three new software/FPGA components. This is detailed in the sections below.

- **libsidekiq:** This is the primary library that supports the use of a separate external transport implementation. The existing PCIe and USB transport layers used by libsidekiq are already bundled with libsidekiq and are available for use by applications. Applications that wish to use a custom transport implementation may enhance their application by registering the custom transport implementation (skiq_register_custom_transport()) and initializing libsidekiq to use that custom transport (skiq_init_xport()).
  - `int32_t skiq_register_custom_transport(skiq_xport_card_functions_t * functions)` - This function performs registration of all the required card function pointers so that libsidekiq knows which functions should be called in the custom transport implementation for the required transport operations. Card functions (probe, init, exit) are registered with this function. Probe and init functions can register functions related to FPGA, RX, and TX functionality depending on a custom transport's capability. This includes functions for reading/writing FPGA registers, starting/stopping streaming, receiving contiguous blocks of samples, and flushing the transport. See the Sidekiq Custom Transport Implementation Guide section in Sidekiq API documentation for more details.

  - `int32_t skiq_init ( skiq_xport_type_t type, skiq_xport_init_level_t level, uint8_t * p_card_nums, uint8_t num_cards )`
  - This function is used to initialize the library and cards. If using the custom transport implementation, the transport functions must be registered prior to calling this API. The specified level may be ' basic' where only FPGA related functions are registered, or 'full'where RX / TX streaming related functions are registered. See the Sidekiq API documentation for more details on this function and related enumerations.

- **Custom SW "Driver"**: This is the Linux kernel space driver that may be required by the custom transport layer to support both register and streaming operations with hardware. The assumption here is that the custom transport implementation calls this software interface to provide the physical transport between the CPU and the FPGA.

- **Custom FPGA Interface**: This is the FPGA block that manages both the register and streaming interfaces on the FPGA. The assumption here is that the vast majority of the stock Sidekiq FPGA reference design will be preserved, replacing only the PCIe/DMA interface currently provided by Northwest Logic with the custom transport interface.

7.3 Custom Transport Implementation Guide

7.3.1 Overview

The current implementation of custom FPGA transport support has the ability to perform register transactions as well as receive sample streaming. While the transmit sample streaming hooks are in place, a custom transport implementation may leave them at their defaults. This release is based on libsidekiq v3.1.0 and adds the custom FPGA transport support and is planned to be released as libsidekiq v3.2.0.

7.3.2 Transport Implementation

In the SDK's top-level directory there are two new directories that have example implementations. The first, **custom_xport_bare**, is a simple stub example where all functions print their parameters and return success. This example can be used as a basis for new work. The second example, **custom_xport_example**, is the PCIe transport layer implementation and can be compiled and referenced as a complete example.

7.3.3 Compiling the Transport Implementation

1. Change directory to either **custom_xport_bare** or **custom_xport_example**
2. Type `make BUILD_CONFIG=x86_64.gcc`

This builds the custom transport and some example test applications. A test application's object file is linked with the custom transport object(s) and libsidekiq's static library. The example test applications are the same as their test_apps counterpart with the exception of calling skiq_register_custom_transport(), and specifying skiq_xport_type_custom in calls to skiq_init().

7.3.4 API

Below is a summary of the function sets that should be supported by the transport library.

7.3.5 Card Functions

All three card functions are required for a custom transport implementation. Pointers to the functions are collected into a skiq_xport_card_functions_t struct and passed to skiq_register_custom_transport() function before calling skiq_init_xport().

- `int32_t (*card_probe)( uint64_t *p_uid_list, uint8_t *p_num_cards );`
- `int32_t (*card_init)( skiq_xport_init_level_t level, uint64_t xport_uid );`
- `int32_t (*card_exit)( skiq_xport_init_level_t level, uint64_t xport_uid );`

The card_probe and card_init are responsible for further registering the remaining transport function sets (FPGA, RX, and TX) based on the caller's skiq_xport_init_level_t request and the card's capabilities. Refer to the **custom_xport_bare** example.

7.3.6 FPGA Functions

The FPGA functions implement transporting requests to read / write registers as well as bringing up / down the transport link to the FPGA (for reprogramming). Pointers to the functions are collected into a skiq_xport_fpga_functions_t struct and registered on a per-card basis by calling xport_register_fpga_functions().

- `int32_t (fpga_reg_read)( uint64_t xport_uid, uint32_t addr, uint32_t p_data );`
- `int32_t (*fpga_reg_write)( uint64_t xport_uid, uint32_t addr, uint32_t data );`
- `int32_t (*fpga_down)( uint64_t xport_uid );`
- `int32_t (*fpga_up)( uint64_t xport_uid );`

7.3.7 RX Functions

The RX functions implement preparing the transport link for starting, stopping, pausing, and resuming receive sample data streaming as well as flushing buffered receive data and transporting receive sample data. Pointers to the functions are collected into a skiq_xport_rx_functions_t struct and registered on a per-card basis by calling xport_register_rx_functions().

- `int32_t (*rx_start_streaming)( uint64_t xport_uid, skiq_rx_hdl_t hdl );`
- `int32_t (*rx_stop_streaming)( uint64_t xport_uid, skiq_rx_hdl_t hdl );`
- `int32_t (*rx_pause_streaming)( uint64_t xport_uid );`
- `int32_t (*rx_resume_streaming)( uint64_t xport_uid );`
- `int32_t (*rx_flush)( uint64_t xport_uid );`
- `int32_t (*rx_receive)( uint64_t xport_uid, uint8_t **pp_data, uint32_t *p_data_len );`

7.3.8 TX Functions

The TX functions implement preparing the transport link for starting and stopping transmit sample data as well as transporting transmit sample data. Pointers to the functions are collected into a skiq_xport_tx_-functions_t struct and registered on a per-card basis by calling xport_register_tx_functions().

- `int32_t (*tx_initialize)( uint64_t xport_uid, skiq_tx_transfer_mode_t tx_transfer_mode, uint32_t num_bytes_to_send, uint8_t num_send_threads, void (*tx_complete_cb)(int32_t status, uint32_t *p_data) );`
- `int32_t (*tx_start_streaming)( uint64_t xport_uid, skiq_tx_hdl_t hdl );`
- `int32_t (*tx_pre_stop_streaming)( uint64_t xport_uid, skiq_tx_hdl_t hdl );`
- `int32_t (*tx_stop_streaming)( uint64_t xport_uid, skiq_tx_hdl_t hdl );`
- `int32_t (*tx_transmit)( uint64_t xport_uid, skiq_tx_hdl_t hdl, int32_t *p_samples, void *p_private );`

Note

It is NOT recommended to use xport_register_fpga_functions(), xport_register_rx_functions(), xport_-register_tx_functions(), xport_unregister_fpga_functions(), xport_unregister_rx_functions(), or xport-_unregister_tx_functions() from functions other than the card_init and card_exit implementations.

Chapter 8

Timestamp Slips within AD9361 Products

8.1 Overview

Products that use the AD9361 RFIC will have timestamp slips when using API functions that need to deactivate the sample clock in order to make updates to the radio configuration.

This occurs when:

- updating the LO frequency
- updating the sample rate
- running the transmit quadrature calibration

Functions that will affect the timestamp:

- skiq_write_rx_LO_freq()
- skiq_write_rx_sample_rate_and_bandwidth()
- skiq_write_tx_LO_freq()
- skiq_run_tx_quadcal()
- skiq_write_rx_freq_tune_mode()
- skiq_write_tx_freq_tune_mode()

Functions that will be affected by the timestamp slip:

- skiq_read_last_1pps_timestamp()
- skiq_receive()
- skiq_transmit()
- skiq_read_curr_rx_timestamp()
- skiq_read_curr_tx_timestamp()

It is recommended to use the system clock - which is not subject to interruptions - if a consistent time source is needed.

Chapter 9

Class Index

9.1 Class List

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

Chapter 10

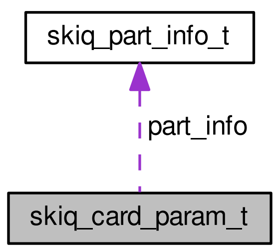
Class Documentation

10.1 skiq_card_param_t Struct Reference

Parameters related to a physical Sidekiq card.

`#include <sidekiq_params.h>`

Collaboration diagram for skiq_card_param_t:



Public Attributes

- skiq_xport_init_level_t init_level
- skiq_part_t part_type
- skiq_fmc_carrier_t part_fmc_carrier
- skiq_part_info_t part_info
- skiq_xport_type_t xport
- bool is_accelerometer_present
- uint8_t card
- char serial_string [SKIQ_SERIAL_NUM_STRLEN]

10.1.1 Detailed Description

Parameters related to a physical Sidekiq card.

Definition at line 27 of file sidekiq_params.h.

10.1.2 Member Data Documentation

10.1.2.1 skiq_xport_init_level_t init_level

The initialization level of a given card.

Definition at line 29 of file sidekiq_params.h.

10.1.2.2 skiq_part_t part_type

The Sidekiq's part type (e.x. "mPCIe", "M.2", "X2", etc).

Definition at line 32 of file sidekiq_params.h.

10.1.2.3 skiq_fmc_carrier_t part_fmc_carrier

The Sidekiq platform's detected FMC carrier (if applicable)

Definition at line 35 of file sidekiq_params.h.

10.1.2.4 skiq_part_info_t part_info

Vendor information related to a given part and its configuration.

Definition at line 38 of file sidekiq_params.h.

10.1.2.5 skiq_xport_type_t xport

Transport configuration for the Sidekiq card (e.x. "PCIe", "USB", or "custom").

Definition at line 41 of file sidekiq_params.h.

10.1.2.6 bool is_accelerometer_present

Boolean used to indicate if the accelerometer is physically present.

Definition at line 44 of file sidekiq_params.h.

10.1.2.7 uint8_t card

Card identifier used for API calls.

Definition at line 47 of file sidekiq_params.h.

10.1.2.8 char serial_string[SKIQ_SERIAL_NUM_STRLEN]

String representation of the serial number of the Sidekiq card.

Definition at line 50 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.2 skiq_fpga_param_t Struct Reference

Parameter for the Sidekiq's on board FPGA.

`#include <sidekiq_params.h>`

Public Attributes

- skiq_fpga_device_t fpga_device
- skiq_fpga_tx_fifo_size_t tx_fifo_size
- uint32_t build_date
- uint32_t git_hash
- uint32_t baseline_hash
- uint64_t sys_timestamp_freq
- uint8_t version_major
- uint8_t version_minor
- uint8_t version_patch
- fpga_state_t fpga_state

10.2.1 Detailed Description

Parameter for the Sidekiq's on board FPGA.

Definition at line 65 of file sidekiq_params.h.

10.2.2 Member Data Documentation

10.2.2.1 skiq_fpga_device_t fpga_device

The Sidekiq platform's FPGA device (may vary with FMC carrier)

See Also

skiq_card_param_t
skiq_fmc_carrier_t

Definition at line 67 of file sidekiq_params.h.

10.2.2.2 skiq_fpga_tx_fifo_size_t tx_fifo_size

Enumerated value of the Tx FIFO depth on the FPGA.

Definition at line 75 of file sidekiq_params.h.

10.2.2.3 uint32_t build_date

Date that the FPGA image was build (YYMMDDHH).

Definition at line 78 of file sidekiq_params.h.

10.2.2.4 uint32_t git_hash

Git commit hash of the FPGA build.

Definition at line 81 of file sidekiq_params.h.

10.2.2.5 uint32_t baseline_hash

Git commit hash of the FPGA build as delivered by Epiq to the user. If the user makes changes to the FPGA and rebuilds, then the git_hash will change and the baseline_hash will remain the same.

Definition at line 84 of file sidekiq_params.h.

10.2.2.6 uint64_t sys_timestamp_freq

Frequency at what the system timestamp runs in hertz.

Definition at line 88 of file sidekiq_params.h.

10.2.2.7 uint8_t version_major

Major version of the FPGA release.

Definition at line 91 of file sidekiq_params.h.

10.2.2.8 uint8_t version_minor

Minor version of the FPGA release.

Definition at line 94 of file sidekiq_params.h.

10.2.2.9 uint8_t version_patch

Patch version of the FPGA release (available in FPGA bitstreams with version 3.8 or later, =0 otherwise)

Definition at line 97 of file sidekiq_params.h.

10.2.2.10 fpga_state_t fpga_state

State of the running fpga version

Definition at line 100 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.3 skiq_fw_param_t Struct Reference

Parameters for the firmware loaded onto a Sidekiq.

`#include <sidekiq_params.h>`

Public Attributes

- bool is_present
- uint16_t enumeration_delay_ms
- uint8_t version_major
- uint8_t version_minor

10.3.1 Detailed Description

Parameters for the firmware loaded onto a Sidekiq.

Definition at line 106 of file sidekiq_params.h.

10.3.2 Member Data Documentation

10.3.2.1 bool is_present

Boolean indicating if firmware is present or absent on the Sidekiq.

Definition at line 108 of file sidekiq_params.h.

10.3.2.2 uint16_t enumeration_delay_ms

Delay in milliseconds which firmware waits before enumerating on the USB bus.

Definition at line 111 of file sidekiq_params.h.

10.3.2.3 uint8_t version_major

Major version of the firmware release.

Definition at line 115 of file sidekiq_params.h.

10.3.2.4 uint8_t version_minor

Minor version of the firmware release.

Definition at line 118 of file sidekiq_params.h.

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

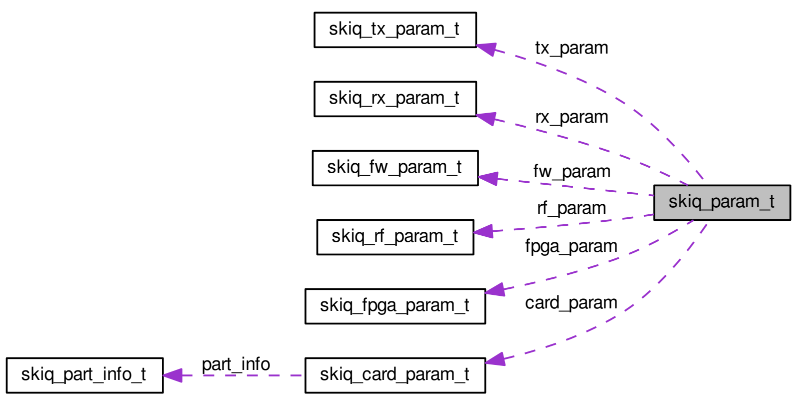
- sidekiq_core/inc/sidekiq_params.h

10.4 skiq_param_t Struct Reference

Parameters for the entire Sidekiq.

`#include <sidekiq_params.h>`

Collaboration diagram for skiq_param_t:



Public Attributes

- skiq_card_param_t card_param
- skiq_fpga_param_t fpga_param
- skiq_fw_param_t fw_param
- skiq_rf_param_t rf_param
- skiq_rx_param_t rx_param [skiq_rx_hdl_end]
- skiq_tx_param_t tx_param [skiq_tx_hdl_end]

10.4.1 Detailed Description

Parameters for the entire Sidekiq.

Note

Must be initialized to skiq_xport_init_level_full to access all members of this struct.

Definition at line 289 of file sidekiq_params.h.

10.4.2 Member Data Documentation

10.4.2.1 skiq_card_param_t card_param

Definition at line 291 of file sidekiq_params.h.

10.4.2.2 skiq_fpga_param_t fpga_param

Definition at line 292 of file sidekiq_params.h.

10.4.2.3 skiq_fw_param_t fw_param

Definition at line 293 of file sidekiq_params.h.

10.4.2.4 skiq_rf_param_t rf_param

Definition at line 294 of file sidekiq_params.h.

10.4.2.5 skiq_rx_param_t rx_param[skiq_rx_hdl_end]

Definition at line 295 of file sidekiq_params.h.

10.4.2.6 skiq_tx_param_t tx_param[skiq_tx_hdl_end]

Definition at line 296 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.5 skiq_part_info_t Struct Reference

Sidekiq Part Information.

`#include <sidekiq_types.h>`

Public Attributes

- char number_string [SKIQ_PART_NUM_STRLEN]
- char revision_string [SKIQ_REVISION_STRLEN]
- char variant_string [SKIQ_VARIANT_STRLEN]

10.5.1 Detailed Description

Sidekiq Part Information.

Definition at line 606 of file sidekiq_types.h.

10.5.2 Member Data Documentation

10.5.2.1 char number_string[SKIQ_PART_NUM_STRLEN]

Definition at line 608 of file sidekiq_types.h.

10.5.2.2 char revision_string[SKIQ_REVISION_STRLEN]

Definition at line 609 of file sidekiq_types.h.

10.5.2.3 char variant_string[SKIQ_VARIANT_STRLEN]

Definition at line 610 of file sidekiq_types.h.

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

- sidekiq_core/inc/sidekiq_types.h

10.6 skiq_rf_param_t Struct Reference

Parameters for the Sidekiq's RF capabilities.

`#include <sidekiq_params.h>`

Public Attributes

- skiq_ref_clock_select_t ref_clock_config
- bool is_rf_port_fixed
- bool is_rf_port_tdd_supported
- bool is_rf_port_trx_supported
- uint8_t num_rx_channels
- skiq_rx_hdl_t rx_handles [skiq_rx_hdl_end]
- uint8_t num_tx_channels
- skiq_tx_hdl_t tx_handles [skiq_tx_hdl_end]
- uint32_t ref_clock_freq
- uint16_t warp_value_max
- uint16_t warp_value_min
- float warp_value_unit

10.6.1 Detailed Description

Parameters for the Sidekiq's RF capabilities.

Note

Must be initialized to skiq_xport_init_level_full to have access to certain members of this struct.

Definition at line 126 of file sidekiq_params.h.

10.6.2 Member Data Documentation

10.6.2.1 skiq_ref_clock_select_t ref_clock_config

Enumerated value of the Sidekiq's reference clock configuration.

Definition at line 128 of file sidekiq_params.h.

10.6.2.2 bool is_rf_port_fixed

Boolean indicating if the RF ports can or can not be configured dynamically.

Definition at line 131 of file sidekiq_params.h.

10.6.2.3 bool is_rf_port_tdd_supported

Boolean indicating if Time Division Duplex is supported. DEPRECATED: replaced by is_rf_port_trx_supported

Definition at line 135 of file sidekiq_params.h.

10.6.2.4 bool is_rf_port_trx_supported

Boolean indicating if RF ports can be switched between receive/transmit modes

Definition at line 139 of file sidekiq_params.h.

10.6.2.5 uint8_t num_rx_channels

Total number of Rx ports on the Sidekiq. This value can be used to index into the skiq_rx_param_t array of skiq_param_t struct.

Note

Must be initialized to skiq_xport_init_level_full

Definition at line 142 of file sidekiq_params.h.

10.6.2.6 skiq_rx_hdl_t rx_handles[skiq_rx_hdl_end]

List of RX handle(s) on the Sidekiq. This array can be used to look up what handles are valid. The number of valid entries in this array is represented by *num_rx_channels*

Note

The *rx_handles*[] array is indexed by 0 ... num_rx_channels and not by skiq_rx_hdl_t!

Definition at line 147 of file sidekiq_params.h.

10.6.2.7 uint8_t num_tx_channels

Total number of Tx ports on the Sidekiq. This value can be used to index into the skiq_tx_param_t array of skiq_param_t struct.

Note

Must be initialized to skiq_xport_init_level_full

Definition at line 155 of file sidekiq_params.h.

10.6.2.8 skiq_tx_hdl_t tx_handles[skiq_tx_hdl_end]

List of TX handle(s) on the Sidekiq. This array can be used to look up what handles are valid. The number of valid entries in this array is represented by *num_tx_channels*

Note

The *tx_handles*[] array is indexed by 0 ... num_tx_channels and not by skiq_tx_hdl_t!

Definition at line 160 of file sidekiq_params.h.

10.6.2.9 uint32_t ref_clock_freq

The frequency of the reference clock in hertz

Definition at line 168 of file sidekiq_params.h.

10.6.2.10 uint16_t warp_value_max

Maximum value for warp voltage control

Definition at line 171 of file sidekiq_params.h.

10.6.2.11 uint16_t warp_value_min

Minimum value for warp voltage control

Definition at line 174 of file sidekiq_params.h.

10.6.2.12 float warp_value_unit

Approximate number of ppb (parts per billion) per warp value unit

Definition at line 177 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.7 skiq_rx_block_t Struct Reference

Sidekiq Receive Block type definition for use with skiq_receive.

`#include <sidekiq_types.h>`

Public Attributes

- volatile uint64_t hdl:6

  Receive handle (6 bits) indicating the receive handle associated with the received sample block.
- volatile uint64_t overload:1

  RF Overload (1 bit) indicating whether or not the RF input was overloaded for the received sample block.
- volatile uint64_t rfic_control:8

  RFIC control word (8 bits) carries metadata from the RFIC, typically the receive gain index.
- volatile uint64_t id:8

  Channel ID (8 bits) used by channelizer.
- volatile uint64_t system_meta:6

  System metadata (6 bits) (unused / reserved)
- volatile uint64_t version:3

  Packet version field (3 bits)
- volatile uint64_t user_meta:32

  User metadata (32 bits) typically populated by a custom FPGA build.

- volatile uint64_t rf_timestamp

  RF timestamp (64 bits) associated with the received sample block.
- volatile uint64_t sys_timestamp

  System timestamp (64 bits) associated with the received sample block.
- union {
    struct {
      volatile uint64_t hdl:6
        Receive handle (6 bits) indicating the receive handle associated with the received sample block.
      volatile uint64_t overload:1
        RF Overload (1 bit) indicating whether or not the RF input was overloaded for the received sample block.
      volatile uint64_t rfic_control:8
        RFIC control word (8 bits) carries metadata from the RFIC, typically the receive gain index.
      volatile uint64_t id:8
        Channel ID (8 bits) used by channelizer.
      volatile uint64_t system_meta:6
        System metadata (6 bits) (unused / reserved)
      volatile uint64_t version:3
        Packet version field (3 bits)
      volatile uint64_t user_meta:32
        User metadata (32 bits) typically populated by a custom FPGA build.
    }
  };

- volatile int16_t data [ ]

  array of unpacked IQ samples (16 bits per I or Q value). Q0 is data[0], I0 is data[1], Q1 is data[2], I1 is data[3], and so on.

10.7.1 Detailed Description

Sidekiq Receive Block type definition for use with skiq_receive.

Since

Type definition added in **v4.0.0**

See Also

skiq_receive

Definition at line 1125 of file sidekiq_types.h.

10.7.2 Member Data Documentation

10.7.2.1 volatile uint64_t rf_timestamp

RF timestamp (64 bits) associated with the received sample block.

Definition at line 1132 of file sidekiq_types.h.

10.7.2.2 volatile uint64_t sys_timestamp

System timestamp (64 bits) associated with the received sample block.

Definition at line 1134 of file sidekiq_types.h.

10.7.2.3 volatile uint64_t hdl

Receive handle (6 bits) indicating the receive handle associated with the received sample block.

Definition at line 1144 of file sidekiq_types.h.

10.7.2.4 volatile uint64_t overload

RF Overload (1 bit) indicating whether or not the RF input was overloaded for the received sample block.

Definition at line 1147 of file sidekiq_types.h.

10.7.2.5 volatile uint64_t rfic_control

RFIC control word (8 bits) carries metadata from the RFIC, typically the receive gain index.

See Also

skiq_write_rfic_control_output_config

Definition at line 1150 of file sidekiq_types.h.

10.7.2.6 volatile uint64_t id

Channel ID (8 bits) used by channelizer.

Definition at line 1153 of file sidekiq_types.h.

10.7.2.7 volatile uint64_t system_meta

System metadata (6 bits) (unused / reserved)

Definition at line 1154 of file sidekiq_types.h.

10.7.2.8 volatile uint64_t version

Packet version field (3 bits)

Definition at line 1156 of file sidekiq_types.h.

10.7.2.9 volatile uint64_t user_meta

User metadata (32 bits) typically populated by a custom FPGA build.

Definition at line 1157 of file sidekiq_types.h.

10.7.2.10 union { ... }

10.7.2.11 volatile int16_t data[ ]

array of unpacked IQ samples (16 bits per I or Q value). Q0 is *data*[0], I0 is *data*[1], Q1 is *data*[2], I1 is *data*[3], and so on.

Definition at line 1161 of file sidekiq_types.h.

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

- sidekiq_core/inc/sidekiq_types.h

10.8 skiq_rx_param_t Struct Reference

Parameters for each Rx channel on a Sidekiq card.

```
#include <sidekiq_params.h>
```

Public Attributes

- skiq_rx_hdl_t handle
- skiq_filt_t filters [skiq_filt_max]
- uint16_t atten_quarter_db_max
- uint16_t atten_quarter_db_min
- uint8_t gain_index_max
- uint8_t gain_index_min

- uint8_t iq_resolution
- uint64_t lo_freq_max
- uint64_t lo_freq_min
- uint8_t num_filters
- uint32_t sample_rate_max
- uint32_t sample_rate_min
- uint8_t num_fixed_rf_ports
- skiq_rf_port_t fixed_rf_ports [skiq_rf_port_max]
- uint8_t num_trx_rf_ports
- skiq_rf_port_t trx_rf_ports [skiq_rf_port_max]
- uint32_t cal_type_mask

10.8.1 Detailed Description

Parameters for each Rx channel on a Sidekiq card.

Note

Must be initialized to skiq_xport_init_level_full to access any members of this struct

Definition at line 185 of file sidekiq_params.h.

10.8.2 Member Data Documentation

10.8.2.1 skiq_rx_hdl_t handle

Handle associated with this set of RX parameters

Definition at line 187 of file sidekiq_params.h.

10.8.2.2 skiq_filt_t filters[skiq_filt_max]

Filters available for the given Rx channel.

Definition at line 190 of file sidekiq_params.h.

10.8.2.3 uint16_t atten_quarter_db_max

Maximum attenuation in quarter dB steps.

Definition at line 193 of file sidekiq_params.h.

10.8.2.4 uint16_t atten_quarter_db_min

Minimum attenuation in quarter dB steps.

Definition at line 195 of file sidekiq_params.h.

10.8.2.5 uint8_t gain_index_max

Maximum index for gain profile selection.

Definition at line 198 of file sidekiq_params.h.

10.8.2.6 uint8_t gain_index_min

Minimum index for gain profile selection.

Definition at line 201 of file sidekiq_params.h.

10.8.2.7 uint8_t iq_resolution

Number of resolution bits for each I/Q signal, that is, I is N bits and Q is N bits.

Definition at line 204 of file sidekiq_params.h.

10.8.2.8 uint64_t lo_freq_max

Maximum frequency that the LO can be tuned to in hertz.

Definition at line 208 of file sidekiq_params.h.

10.8.2.9 uint64_t lo_freq_min

Minimum frequency that the LO can be tuned to in hertz.

Definition at line 211 of file sidekiq_params.h.

10.8.2.10 uint8_t num_filters

Total number of filters available on the Rx channel.

Definition at line 214 of file sidekiq_params.h.

10.8.2.11 uint32_t sample_rate_max

Maximum rate at which I/Q sample clock can be driven in hertz.

Definition at line 217 of file sidekiq_params.h.

10.8.2.12 uint32_t sample_rate_min

Minimum rate at which I/Q sample clock can be driven in hertz.

Definition at line 220 of file sidekiq_params.h.

10.8.2.13 uint8_t num_fixed_rf_ports

Total number of fixed RX ports

Definition at line 223 of file sidekiq_params.h.

10.8.2.14 skiq_rf_port_t fixed_rf_ports[skiq_rf_port_max]

list of fixed RX RF ports

Definition at line 226 of file sidekiq_params.h.

10.8.2.15 uint8_t num_trx_rf_ports

Total number of TRX ports

Definition at line 229 of file sidekiq_params.h.

10.8.2.16 skiq_rf_port_t trx_rf_ports[skiq_rf_port_max]

list of TRX RF ports

Definition at line 232 of file sidekiq_params.h.

10.8.2.17 uint32_t cal_type_mask

mask of calibration types [skiq_rx_cal_type_t] available

Definition at line 235 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.9 skiq_tx_block_t Struct Reference

Sidekiq Transmit Block type definition for use with skiq_transmit and skiq_tx_callback_t.

`#include <sidekiq_types.h>`

Public Attributes

- uint32_t miscHigh

  high word of metadata (32 bits) (unused)
- uint32_t miscLow

  low word of metadata (32 bits) (unused)
- uint64_t timestamp

  RF timestamp (64 bits) for transmitted block when transmit flow mode is skiq_tx_with_timestamps_data_flow_-mode.
- int16_t data [ ]

  array of unpacked IQ samples (16 bits per I or Q value)

10.9.1 Detailed Description

Sidekiq Transmit Block type definition for use with skiq_transmit and skiq_tx_callback_t.

Since

Type definition added in **v4.0.0**

See Also

skiq_transmit
skiq_tx_callback_t

Definition at line 1078 of file sidekiq_types.h.

10.9.2 Member Data Documentation

10.9.2.1 uint32_t miscHigh

high word of metadata (32 bits) (unused)

Definition at line 1085 of file sidekiq_types.h.

10.9.2.2 uint32_t miscLow

low word of metadata (32 bits) (unused)

Definition at line 1086 of file sidekiq_types.h.

10.9.2.3 uint64_t timestamp

RF timestamp (64 bits) for transmitted block when transmit flow mode is skiq_tx_with_timestamps_data_-flow_mode.

Definition at line 1087 of file sidekiq_types.h.

10.9.2.4 int16_t data[ ]

array of unpacked IQ samples (16 bits per I or Q value)

Definition at line 1088 of file sidekiq_types.h.

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

- sidekiq_core/inc/sidekiq_types.h

10.10 skiq_tx_param_t Struct Reference

Parameters for each Tx channel on a Sidekiq card.

`#include <sidekiq_params.h>`

Public Attributes

- skiq_tx_hdl_t handle

- skiq_filt_t filters [skiq_filt_max]
- uint16_t atten_quarter_db_max
- uint16_t atten_quarter_db_min
- uint8_t iq_resolution
- uint64_t lo_freq_max
- uint64_t lo_freq_min
- uint8_t num_filters
- uint32_t sample_rate_max
- uint32_t sample_rate_min
- uint8_t num_fixed_rf_ports
- skiq_rf_port_t fixed_rf_ports [skiq_rf_port_max]
- uint8_t num_trx_rf_ports
- skiq_rf_port_t trx_rf_ports [skiq_rf_port_max]

10.10.1 Detailed Description

Parameters for each Tx channel on a Sidekiq card.

Note

Must be initialized to skiq_xport_init_level_full to access any members of this struct.

Definition at line 242 of file sidekiq_params.h.

10.10.2 Member Data Documentation

10.10.2.1 skiq_tx_hdl_t handle

Handle associated with this set of TX parameters

Definition at line 244 of file sidekiq_params.h.

10.10.2.2 skiq_filt_t filters[skiq_filt_max]

Filters available for the given Tx channel.

Definition at line 247 of file sidekiq_params.h.

10.10.2.3 uint16_t atten_quarter_db_max

Maximum attenuation in quarter dB steps.

Definition at line 250 of file sidekiq_params.h.

10.10.2.4 uint16_t atten_quarter_db_min

Minimum attenuation in quarter dB steps.

Definition at line 252 of file sidekiq_params.h.

10.10.2.5 uint8_t iq_resolution

Number of resolution bits for each I/Q signal, that is, I is N bits and Q is N bits"

Definition at line 255 of file sidekiq_params.h.

10.10.2.6 uint64_t lo_freq_max

Maximum frequency that the LO can be tuned to in hertz.

Definition at line 259 of file sidekiq_params.h.

10.10.2.7 uint64_t lo_freq_min

Minimum frequency that the LO can be tuned to in hertz.

Definition at line 261 of file sidekiq_params.h.

10.10.2.8 uint8_t num_filters

Total number of filters available on the Tx channel.

Definition at line 264 of file sidekiq_params.h.

10.10.2.9 uint32_t sample_rate_max

Maximum rate at which I/Q sample clock can be driven in hertz.

Definition at line 267 of file sidekiq_params.h.

10.10.2.10 uint32_t sample_rate_min

Minimum rate at which I/Q sample clock can be driven in hertz.

Definition at line 270 of file sidekiq_params.h.

10.10.2.11 uint8_t num_fixed_rf_ports

Total number of fixed TX ports

Definition at line 273 of file sidekiq_params.h.

10.10.2.12 skiq_rf_port_t fixed_rf_ports[skiq_rf_port_max]

list of fixed TX RF ports

Definition at line 276 of file sidekiq_params.h.

10.10.2.13 uint8_t num_trx_rf_ports

Total number of TRX ports

Definition at line 279 of file sidekiq_params.h.

10.10.2.14 skiq_rf_port_t trx_rf_ports[skiq_rf_port_max]

list of TRX RF ports

Definition at line 282 of file sidekiq_params.h.

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

- sidekiq_core/inc/sidekiq_params.h

10.11 skiq_xport_card_functions_t Struct Reference

`#include <sidekiq_xport_types.h>`

Public Attributes

- int32_t(* card_probe)(uint64_t *p_uid_list, uint8_t *p_num_uids)
- int32_t(* card_hotplug)(uint64_t uid_list[ ], uint8_t *p_nr_uids, uint64_t no_probe_uids[ ], uint8_t nr_no_probe_uids)
- int32_t(* card_init)(skiq_xport_init_level_t level, uint64_t xport_uid)
- int32_t(* card_exit)(skiq_xport_init_level_t level, uint64_t xport_uid)
- int32_t(* card_read_priv_data)(uint64_t xport_uid, uint8_t max_num_bytes, uint8_t *p_num_bytes, uint8_t *p_private_data)
- int32_t(* card_write_priv_data)(uint64_t xport_uid, uint8_t num_bytes, uint8_t *p_private_data)

10.11.1 Detailed Description

The skiq_xport_card_functions_t describes a structure of function pointers to be registered for a custom transport using skiq_register_custom_transport(). The .card_probe and .card_init are required functions.

Definition at line 107 of file sidekiq_xport_types.h.

10.11.2 Member Data Documentation

10.11.2.1 int32_t(* card_probe)(uint64_t *p_uid_list, uint8_t *p_num_uids)

card_probe() is called once after a system start-up. After that, card_hotplug() is responsible for updating card presence and/or absence.

The `p_uid_list` and `p_num_cards` pointers are provided by caller.

This function should assign a transport identifier to each transport interface detected. The UID should uniquely identify the transport at the transport layer and cannot be duplicated within the `p_uid_list`.

This function should assign `*p_num_uids` to the number of UIDs discovered during probe with a maximum of SKIQ_MAX_NUM_CARDS.

For example, if there are 3 UIDs discovered during the probe with UIDs of 1, 3, and 2, then

- p_uid_list[0]=1
- p_uid_list[1]=3

- p_uid_list[2]=2
- ∗p_num_uids=3

Parameters

out	*p_uid_list*	points to an array of uint64_t UID values (limited from 0 to SKIQ_MAX_NUM_CARDS - 1) with SKIQ_MAX_NUM_CARDS entries.
out	*p_num_uids*	reference to the number of valid entries in `p_uid_list`, up to SKIQ_MAX_NUM_CARDS

Returns

status where 0=success, anything else is an error.

Definition at line 135 of file sidekiq_xport_types.h.

10.11.2.2 int32_t(∗ card_hotplug)(uint64_t uid_list[ ], uint8_t ∗p_nr_uids, uint64_t no_probe_uids[ ], uint8_t nr_no_probe_uids)

The card_hotplug() function pointer may be called during skiq_init() or any time Sidekiq cards are probed (e.g. after FPGA reconfiguration).

`uid_list` and `p_nr_uids` are provided by the caller and are to be populated by this function while `no_probe_uids` and `nr_no_probe_uids` are provided by the caller as a list of UIDs that are **NOT** permitted to be probed by the transport. These tranport identifiers are in use by another process and can be corrupted if probed.

Attention

This function should **NOT** probe transport identifiers that are listed in the `no_probe_uids` list. This function should **NOT** indicate UIDs from the `no_probe_uids` list as detected in the `uid_list`.

This function should assign a transport identifier to each transport interface detected. The UID should uniquely identify the transport at the transport layer and cannot be duplicated within the `uid_list`.

This function should assign `∗p_nr_uids` to the number of UIDs discovered during probe with a maximum of SKIQ_MAX_NUM_CARDS.

For example, if there are 3 UIDs discovered during the probe with UIDs of 1, 3, and 2, but 3 is listed in the `no_probe_uids` list, then:

- uid_list[0]=1
- uid_list[2]=2
- ∗p_nr_uids=2

Parameters

out	*uid_list*	an array of uint64_t transport UID values with at most SKIQ_MAX_NUM_CARDS entries

out	*p_nr_uids*	reference to the number of valid entries in uid_list array, up to SKIQ_MAX_NUM_CARDS
in	*no_probe_uids*	an array of uint64_t transport UID values with at most SKIQ_MAX_NUM_CARDS entries
in	*nr_no_probe_uids*	number of valid entries in no_probe_uids array, up to SKIQ_MAX_NUM_CARDS

Returns

status where 0=success, anything else is an error.

Definition at line 174 of file sidekiq_xport_types.h.

10.11.2.3 int32_t(∗ card_init)(skiq_xport_init_level_t level, uint64_t xport_uid)

card_init() is called during skiq_init(). This function performs all the necessary initialization on the UID specified. It is also the responsibility of this function to register FPGA, RX, and TX function pointer structs according the specified level and the card's capabilities.

Parameters

in	*level*	init level to which each card should be initialized
in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 191 of file sidekiq_xport_types.h.

10.11.2.4 int32_t(∗ card_exit)(skiq_xport_init_level_t level, uint64_t xport_uid)

card_exit() is called from skiq_exit(). This function performs all steps necessary to shutdown communication with the card hardware specified in the p_card_list array. It is also the responsibility to unregister FPGA, RX, and TX functionality.

Parameters

in	*level*	init level to which each card was previously initialized
in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 206 of file sidekiq_xport_types.h.

10.11.2.5 int32_t(∗ card_read_priv_data)(uint64_t xport_uid, uint8_t max_num_bytes, uint8_t ∗p_num_bytes, uint8_t ∗p_private_data)

Definition at line 210 of file sidekiq_xport_types.h.

10.11.2.6 int32_t(* card_write_priv_data)(uint64_t xport_uid, uint8_t num_bytes, uint8_t *p_private_data)

Definition at line 216 of file sidekiq_xport_types.h.

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

- transport/inc/sidekiq_xport_types.h

10.12 skiq_xport_fpga_functions_t Struct Reference

`#include <sidekiq_xport_types.h>`

Public Attributes

- int32_t(* fpga_reg_read)(uint64_t xport_uid, uint32_t addr, uint32_t *p_data)
- int32_t(* fpga_reg_write)(uint64_t xport_uid, uint32_t addr, uint32_t data)
- int32_t(* fpga_down)(uint64_t xport_uid)
- int32_t(* fpga_down_reload)(uint64_t xport_uid, uint32_t addr)
- int32_t(* fpga_up)(uint64_t xport_uid)
- int32_t(* fpga_reg_verify)(uint64_t xport_uid, uint32_t addr, uint32_t data)
- int32_t(* fpga_reg_write_and_verify)(uint64_t xport_uid, uint32_t addr, uint32_t data)
- int32_t(* fpga_reg_read_64)(uint64_t xport_uid, uint32_t addr, uint64_t *p_data)
- int32_t(* fpga_reg_write_64)(uint64_t xport_uid, uint32_t addr, uint64_t data)

10.12.1 Detailed Description

The skiq_xport_fpga_functions_t describes a structure of function pointers to be registered for a specified card. Registration occurs from the .card_init implementation by calling xport_register_fpga_functions. Clearing registration occurs from the .card_init or .card_exit implementations by calling xport_unregister_fpga_-functions.

Definition at line 230 of file sidekiq_xport_types.h.

10.12.2 Member Data Documentation

10.12.2.1 int32_t(* fpga_reg_read)(uint64_t xport_uid, uint32_t addr, uint32_t *p_data)

fpga_reg_read() is called widely across libsidekiq's implementation. This function's responsibility is to either populate p_data with the contents of the FPGA's register at address *addr* or return a non-zero error code.

Note

> xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*addr*	address of the requested FPGA register
out	*p_data*	reference to a uint32_t in which to store the register's contents

Returns

status where 0=success, anything else is an error.

Definition at line 247 of file sidekiq_xport_types.h.

10.12.2.2 int32_t(∗ fpga_reg_write)(uint64_t xport_uid, uint32_t addr, uint32_t data)

fpga_reg_write() is called widely across libsidekiq's implementation. This function's responsibility is to either write the contents of data to the FPGA's register at address *addr* or return a non-zero error code.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*addr*	address of the destination FPGA register
in	*data*	value to store in the register

Returns

status where 0=success, anything else is an error.

Definition at line 265 of file sidekiq_xport_types.h.

10.12.2.3 int32_t(∗ fpga_down)(uint64_t xport_uid)

fpga_down() is called before libsidekiq needs to "disrupt" the transport link. Currently this only occurs before the FPGA is undergoing re-programming. This function's responsibility is to tear down communications (transport) with the specified card in preparation for re-programming.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 283 of file sidekiq_xport_types.h.

10.12.2.4 int32_t(∗ fpga_down_reload)(uint64_t xport_uid, uint32_t addr)

fpga_down_reload() is called when libsidekiq needs to tear down the current FPGA communications (transport) layer with the intent of reprogramming it with a FPGA image stored in flash memory.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*addr*	the flash address where the FPGA bitstream resides

Returns

status where 0=success, anything else is an error.

Definition at line 298 of file sidekiq_xport_types.h.

10.12.2.5 int32_t(∗ fpga_up)(uint64_t xport_uid)

fpga_up() is called after libsidekiq "disrupts" the transport link to the specified card. Currently this only occurs after the FPGA has been re-programmed. This function's responsibility is to bring communications (transport) back up with the specified card.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 314 of file sidekiq_xport_types.h.

10.12.2.6 int32_t(∗ fpga_reg_verify)(uint64_t xport_uid, uint32_t addr, uint32_t data)

fpga_reg_verify() is called to verify that the specified address contains value specified by data.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*addr*	register address to be verified
in	*data*	value to be verified

Returns

status where 0=success, anything else is an error.

Definition at line 328 of file sidekiq_xport_types.h.

10.12.2.7 int32_t(∗ fpga_reg_write_and_verify)(uint64_t xport_uid, uint32_t addr, uint32_t data)

fpga_reg_write_and_verify() is called to write a the value specified by data to the register address and verify that the data was written successfully.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*addr*	register address to be verified
in	*data*	value to be verified

Returns

status where 0=success, anything else is an error.

Definition at line 342 of file sidekiq_xport_types.h.

10.12.2.8 int32_t(∗ fpga_reg_read_64)(uint64_t xport_uid, uint32_t addr, uint64_t ∗p_data)

Definition at line 344 of file sidekiq_xport_types.h.

10.12.2.9 int32_t(∗ fpga_reg_write_64)(uint64_t xport_uid, uint32_t addr, uint64_t data)

Definition at line 345 of file sidekiq_xport_types.h.

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

- transport/inc/sidekiq_xport_types.h

10.13 skiq_xport_id_t Struct Reference

```
#include <sidekiq_xport_types.h>
```

Public Attributes

- uint64_t xport_uid
- skiq_xport_type_t type

10.13.1 Detailed Description

Definition at line 94 of file sidekiq_xport_types.h.

10.13.2 Member Data Documentation

10.13.2.1 uint64_t xport_uid

Definition at line 96 of file sidekiq_xport_types.h.

10.13.2.2 skiq_xport_type_t type

Definition at line 97 of file sidekiq_xport_types.h.

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

- transport/inc/sidekiq_xport_types.h

10.14 skiq_xport_rx_functions_t Struct Reference

```
#include <sidekiq_xport_types.h>
```

Public Attributes

- int32_t(∗ rx_configure)(uint64_t xport_uid, uint32_t aggregate_data_rate)
- int32_t(∗ rx_set_block_size)(uint64_t xport_uid, uint32_t block_size)
- int32_t(∗ rx_set_buffered)(uint64_t xport_uid, bool buffered)
- int32_t(∗ rx_start_streaming)(uint64_t xport_uid, skiq_rx_hdl_t hdl)
- int32_t(∗ rx_stop_streaming)(uint64_t xport_uid, skiq_rx_hdl_t hdl)
- int32_t(∗ rx_pause_streaming)(uint64_t xport_uid)
- int32_t(∗ rx_resume_streaming)(uint64_t xport_uid)
- int32_t(∗ rx_flush)(uint64_t xport_uid)
- int32_t(∗ rx_set_transfer_timeout)(uint64_t xport_uid, const int32_t timeout_us)
- int32_t(∗ rx_receive)(uint64_t xport_uid, uint8_t ∗∗pp_data, uint32_t ∗p_data_len)

10.14.1 Detailed Description

The skiq_xport_rx_functions_t describes a structure of function pointers to be registered for a specified card. Registration occurs from the .card_init implementation by calling xport_register_rx_functions. Clearing registration occurs from the .card_init or .card_exit implementations by calling xport_unregister_rx_functions.

Definition at line 357 of file sidekiq_xport_types.h.

10.14.2 Member Data Documentation

10.14.2.1 int32_t(* rx_configure)(uint64_t xport_uid, uint32_t aggregate_data_rate)

rx_configure() is called from skiq_init(), skiq_start_rx_streaming(), skiq_start_rx_streaming_on_1pps(), and skiq_write_rx_sample_rate_and_bandwidth(). This function's responsibility is to inform the transport implementation of the raw data rate (in bytes per second) of the receive stream in case it needs to adjust any transport layer configuration.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*aggregate_data_rate*	raw date rate in bytes per second for the receive IQ stream

Returns

status where 0=success, anything else is an error.

Definition at line 375 of file sidekiq_xport_types.h.

10.14.2.2 int32_t(* rx_set_block_size)(uint64_t xport_uid, uint32_t block_size)

rx_set_block_size() is called from skiq_init(), skiq_start_rx_streaming(), and skiq_start_rx_streaming_on_1pps(). This function's responsibility is to inform the transport implementation of desired the receive block size. Currently the two possible settings would be 4096 bytes (legacy / high throughput) and 256 bytes (low latency).

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*block_size*	desired block size in bytes, applies to all receive handles

Returns

status where 0=success, anything else is an error.

Definition at line 394 of file sidekiq_xport_types.h.

10.14.2.3 int32_t(* rx_set_buffered)(uint64_t xport_uid, bool buffered)

rx_set_buffered() is called from skiq_init(), skiq_start_rx_streaming(), and skiq_start_rx_streaming_on_1pps(). This function's responsibility is to inform the transport implementation of whether packet requests should be buffered (i.e. multiple receive packets should be requested with a single transaction)

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*buffered*	indicates whether the transport packet request should buffer

Returns

status where 0=success, anything else is an error.

Definition at line 412 of file sidekiq_xport_types.h.

10.14.2.4 int32_t(∗ rx_start_streaming)(uint64_t xport_uid, skiq_rx_hdl_t hdl)

rx_start_streaming() is called from skiq_start_rx_streaming() and skiq_start_rx_streaming_on_1pps(). This function's responsibility is perform actions necessary to start retrieving IQ samples from the specified card and handle over the transport link.

Note

This function is called by libsidekiq BEFORE the FPGA is commanded to start collecting samples. libsidekiq's skiq_start_rx_streaming() calls transport RX functions in the following order:

- rx_pause_streaming
- rx_resume_streaming
- rx_flush
- rx_start_streaming

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

hdl will always be a valid skiq_rx_hdl_t

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	handle identifier to prepare the receive IQ stream

Returns

status where 0=success, anything else is an error.

Definition at line 441 of file sidekiq_xport_types.h.

10.14.2.5 int32_t(∗ rx_stop_streaming)(uint64_t xport_uid, skiq_rx_hdl_t hdl)

rx_stop_streaming() is called from skiq_stop_rx_streaming(). This function's responsibility is perform actions necessary to stop retrieving IQ samples from the specified card and handle over the transport link.

Note

This function is called by libsidekiq BEFORE the FPGA is commanded to stop collecting samples.
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

hdl will always be either a valid skiq_rx_hdl_t

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	identifies a handle to halt the receive IQ stream

Returns

status where 0=success, anything else is an error.

Definition at line 462 of file sidekiq_xport_types.h.

10.14.2.6 int32_t(∗ rx_pause_streaming)(uint64_t xport_uid)

rx_pause_streaming() is called from skiq_start_rx_streaming() and skiq_write_rx_sample_rate_and_bandwidth() to signal the transport link to freeze retrieving IQ samples from the FPGA. For some transport links, this function may be NOP'd or assigned NULL

Note

This function is called by libsidekiq BEFORE the FPGA is commanded to start collecting samples. Refer to the note in rx_start_streaming for the call order of transport RX functions.
This function is called by libsidekiq AFTER the RX sample rate is updated. libsikdeiq's skiq_write_rx_sample_rate_and_bandwidth() calls transport functions in the following order:

- rx_pause_streaming
- rx_resume_streaming
- rx_flush

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 489 of file sidekiq_xport_types.h.

10.14.2.7 int32_t(∗ rx_resume_streaming)(uint64_t xport_uid)

rx_resume_streaming() is called from skiq_start_rx_streaming() and skiq_write_rx_sample_rate_and_bandwidth() to signal the transport link to continue retrieving IQ samples from the FPGA. For some transport links, this function may be NOP'd or assigned NULL.

Note

This function is called by libsidekiq BEFORE the FPGA is commanded to start collecting samples. Refer to the note in rx_start_streaming for the call order of transport RX functions in skiq_start_rx_streaming().
This function is called by libsidekiq AFTER the RX sample rate is updated. Refer to the note in rx_pause_streaming for the call order of transport RX functions in skiq_write_rx_sample_rate_and_bandwidth().
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 512 of file sidekiq_xport_types.h.

10.14.2.8 int32_t(∗ rx_flush)(uint64_t xport_uid)

rx_flush() is called from skiq_start_rx_streaming() and skiq_write_rx_sample_rate_and_bandwidth() to signal the transport layer to dump any data buffered while retrieving IQ samples from the FPGA. This function is used internally to flush "stale data" after a call to rx_pause_streaming().

Note

This function is called by libsidekiq BEFORE the FPGA is commanded to start collecting samples. Refer to the note in rx_start_streaming for the call order of transport RX functions in skiq_start_rx_streaming().
This function is called by libsidekiq AFTER the RX sample rate is updated. Refer to the note in rx_pause_streaming for the call order of transport RX functions in skiq_write_rx_sample_rate_and_bandwidth().
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer

Returns

status where 0=success, anything else is an error.

Definition at line 536 of file sidekiq_xport_types.h.

10.14.2.9 int32_t(∗ rx_set_transfer_timeout)(uint64_t xport_uid, const int32_t timeout_us)

rx_set_transfer_timeout() is called from skiq_set_rx_transfer_timeout(), skiq_start_rx_streaming(), skiq_start_rx_streaming_on_1pps(), and skiq_write_rx_sample_rate_and_bandwidth() and is responsible for updating the current receive transfer timeout for the provided card. The currently permissible range of timeout is RX_TRANSFER_WAIT_FOREVER, RX_TRANSFER_NO_WAIT, or a value between 20 and 1000000.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*timeout_us*	minimum timeout in microseconds for a blocking skiq_receive. can be RX_TRANSFER_WAIT_FOREVER, RX_TRANSFER_NO_WAIT, or 20-1000000.

Returns

int32_t status where 0=success, anything else is an error.

Definition at line 554 of file sidekiq_xport_types.h.

10.14.2.10 int32_t(∗ rx_receive)(uint64_t xport_uid, uint8_t ∗∗pp_data, uint32_t ∗p_data_len)

rx_receive() is called from skiq_receive() and is responsible for providing a reference to a block of IQ data of length SKIQ_MAX_RX_BLOCK_SIZE_IN_BYTES and setting ∗p_data_len to SKIQ_MAX_RX_BLOCK_SIZE-_IN_BYTES.

Note

xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
out	*pp_data*	reference to IQ data memory pointer
out	*p_data_len*	reference to value SKIQ_MAX_RX_BLOCK_SIZE_IN_BYTES

Returns

status where 0=success, anything else is an error.

Definition at line 571 of file sidekiq_xport_types.h.

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

- transport/inc/sidekiq_xport_types.h

10.15 skiq_xport_tx_functions_t Struct Reference

```
#include <sidekiq_xport_types.h>
```

Public Attributes

- int32_t(∗ tx_initialize)(uint64_t xport_uid, skiq_tx_transfer_mode_t tx_transfer_mode, uint32_t num-_bytes_to_send, uint8_t num_send_threads, int32_t priority, skiq_tx_callback_t tx_complete_cb)
- int32_t(∗ tx_start_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)
- int32_t(∗ tx_pre_stop_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)
- int32_t(∗ tx_stop_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)
- int32_t(∗ tx_transmit)(uint64_t xport_uid, skiq_tx_hdl_t hdl, int32_t ∗p_samples, void ∗p_private)

10.15.1 Detailed Description

The skiq_xport_tx_functions_t describes a structure of function pointers to be registered for a specified card. Registration occurs from the .card_init implementation by calling xport_register_tx_functions. Clearing registration occurs from the .card_init or .card_exit implementations by calling xport_unregister_tx_functions.

Definition at line 585 of file sidekiq_xport_types.h.

10.15.2 Member Data Documentation

10.15.2.1 int32_t(∗ tx_initialize)(uint64_t xport_uid, skiq_tx_transfer_mode_t tx_transfer_mode, uint32_t num_bytes_to_send, uint8_t num_send_threads, int32_t priority, skiq_tx_callback_t tx_complete_cb)

tx_initialize() is called from skiq_start_tx_streaming() and skiq_start_tx_streaming_on_1pps() and is responsible initializing the transmit parameters.

Note

The skiq_tx_transfer_mode_sync setting should not use threads. The skiq_tx_transfer_mode_async should create *num_send_threads* number of threads for use in an asynchronous mode. The callback tx_complete_cb function is called in async mode when the relevant sample block has been committed. Threads should be torn down in a call to .tx_stop_streaming().
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

tx_transfer_mode will always be either *skiq_tx_transfer_mode_sync* or *skiq_tx_transfer_mode_async*

num_bytes_to_send will always be a multiple of 1024 bytes

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*tx_transfer_-mode*	desired transfer mode - sync or async
in	*num_bytes_to_-send*	number of bytes to expect in each tx_transmit call
in	*num_send_-threads*	number of threads to make available for transmission - value only valid if tx_transfer_mode == skiq_tx_transfer_mode_async
in	*tx_complete_cb*	function to call when transmit block has been committed to the FPGA - value only valid if tx_transfer_mode == skiq_tx_transfer_mode_async

Returns

status where 0=success, anything else is an error.

Definition at line 616 of file sidekiq_xport_types.h.

10.15.2.2 int32_t(∗ tx_start_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)

tx_start_streaming() is called skiq_start_tx_streaming() and skiq_start_tx_streaming_on_1pps() and is responsible for performing steps to prepare the transport link for transmit sample data.

Note

This function is called AFTER the FPGA is commanded that it will be transmitting samples
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

hdl should be ignored

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	identifies a handle to prep the transport link for transmission - should be ignored, retained for legacy purposes

Returns

status where 0=success, anything else is an error.

Definition at line 641 of file sidekiq_xport_types.h.

10.15.2.3 int32_t(∗ tx_pre_stop_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)

tx_pre_stop_streaming() is called from skiq_stop_tx_streaming() and skiq_stop_tx_streaming_on_1pps() and is responsible for performing steps to prepare the transport link to stop transmitting sample data.

Note

This function is called BEFORE the FPGA is commanded to stop transmitting samples
Threads created as part of skiq_tx_transfer_mode_async mode should be destroyed here.
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

hdl should be ignored

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	identifies a handle to halt the transport link for transmission - should be ignored, retained for legacy purposes

Returns

status where 0=success, anything else is an error.

Definition at line 665 of file sidekiq_xport_types.h.

10.15.2.4 int32_t(∗ tx_stop_streaming)(uint64_t xport_uid, skiq_tx_hdl_t hdl)

tx_stop_streaming() is called from skiq_stop_tx_streaming() and skiq_stop_tx_streaming_on_1pps() and is responsible for performing steps to halt the transport link for transmit sample data.

Note

This function is called AFTER the FPGA is commanded to stop transmitting samples
Threads created as part of skiq_tx_transfer_mode_async mode should be destroyed here.
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

hdl should be ignored

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	identifies a handle to halt the transport link for transmission - should be ignored, retained for legacy purposes

Returns

status where 0=success, anything else is an error.

Definition at line 690 of file sidekiq_xport_types.h.

10.15.2.5 int32_t(∗ tx_transmit)(uint64_t xport_uid, skiq_tx_hdl_t hdl, int32_t ∗p_samples, void ∗p_private)

tx_transmit() is called from skiq_tx_transmit() and is responsible for committing sample data to the FPGA over the transport link either in a synchronous or asynchronous manner.

Note

It is required that if transmit was initialized as *skiq_tx_transfer_mode_sync*, that this function blocks until the transmit data is received by the FPGA over the transport link. If transmit was initialized as *skiq_tx_transfer_mode_async*, the function immediately accepts the sample block (as buffer space allows).
xport_uid is the UID that is provided in the init function and the card is "active" (has gone through initialization)

Parameters

in	*xport_uid*	unique ID used to identifer the card at the transport layer
in	*hdl*	identifies a handle for sample data transmission
in	*p_samples*	reference to sample data of length num_bytes_send (from tx_initialize)

Returns

status where 0=success, anything else is an error.

Definition at line 713 of file sidekiq_xport_types.h.

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

- transport/inc/sidekiq_xport_types.h

Index