Application Note

74-0044-170910

Using the ThinkRF R5500 Real-Time Spectrum Analyzer with C++

The ThinkRF C++ API enables the ThinkRF Real Time Spectrum Analyzer (RTSA) to be quickly and easily integrated into your existing or new C++ based applications.

The C++ programming language is widely used for application development. By controlling the ThinkRF RTSA via the C++ API, simple scripts can be used for data acquisition as well as application development in complex systems.

The ThinkRF C++ API also includes several DSP functions which allows signal analysis to be performed at a higher level of abstraction thereby simplifying the implementation of signal processing functions such as spectral data computation, channel power calculation, peak find, and occupied bandwidth.

This application note will walk you through the ThinkRF provided C++ API functions as well as running an example.

Contents

Required Software

To use the ThinkRF C++ API described in this Application Note, the following software is required:

- Windows 7/8/10 32-bit/64-bit operating system;
- Visual Studio 2010 Express;
- RTSA Software and Firmware Release Package. The Release Package may be download from http://www.thinkrf.com/firmware-updates/.

Note: The C++ API can also be used in other environments such as Linux and Mac OS. Examples will be added in the future.

The ThinkRF Real-Time Spectrum Analyzer

This Application Note also assumes you have access and are connected to a ThinkRF Real-Time Spectrum Analyzer (RTSA).

The RTSA is designed for distributed or remote deployment and as such you can use and/or evaluate any of our third-party software applications by connecting to one of ThinkRF's RTSAs which we have deployed on the internet. You may access to one of our evaluation units on the internet from www.thinkrf.com/demo

The RTSA C++ API

The RTSA C++ API is built using the RTSA C-API. The source code for the RTSA C++ API as well as some illustrative examples can be find in the "*\APIs\C++*" directory of the RTSA Software and Firmware Release Package. The "*include*" directory includes all of the header files, and the "*src*" directory includes all of the source code for the C++ API. Compiled windows libraries (*.lib* and *.dll*) are available in the same "*\APIs\C++*" directory.

The following is a list of the functions in the C++ API that can be used to control and acquire data from the ThinkRF RTSA.

Note: Variables with a "*" symbol indicate the variable is passed by reference.

wsaConnect:

Establish a connection to the RTSA.

Input Variables:

- *wsa_handle (64-bit integer): Stores the RTSA socket session information. This handle will be needed in all other API functions to control, and to acquire data.
- *ip (char array): The RTSA's IP address.

Returns (16-bit integer):

- Returns an error code which indicates whether an error has occurred (negative values indicates an error). Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaDisconnect:

Close an established connection to the RTSA.

Input Variables:

- *wsa_handle (64-bit integer by value): The RTSA connection to close.

Returns (16-bit integer):

- Returns an error code which indicates whether an error has occurred (negative values indicates an error). Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaReadData:

Read one IF data packet with all of the corresponding context packets. Note that the application calling this function must allocate memory for the i16_data, q16_data, and i32_data buffers.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- *i16_data (16-bit data buffer): A data buffer to store the 16-bit I-data. The value in this buffer will depend on the RFE mode of the RTSA.
- *q16_data (16-bit data buffer): A data buffer to store the 16-bit Q-data. The value in this buffer will depend on the RFE mode of the RTSA.
- *i32_data (32-bit data buffer): A data buffer to store the 32-bit I-data. This buffer will only contain data if the RTSA is set to 'HDR' RFE mode.
- timeout (unsigned 32-bit integer): How long the read function should read data before timing out.
- *stream_id (unsigned 32-bit integer): The stream id of the IF data packet
- *spectral_inversion (unsigned 8-bit integer): A flag which indicates whether the IF data contains any spectral inversion
- *samples_per_packet (32-bit integer): The number of data samples in the data packet.
- *timestamp_sec (unsigned 32-bit integer): The time in which the data was sampled from the ADC in seconds (UTC format)
- *timestamp_psec (unsigned 32-bit integer): The time in which the data was sampled from the ADC in picoseconds.
- *reference_level (16-bit integer): The reference value used to calibrate the spectral data (in dBm).
- *bandwidth (64-bit integer): The total bandwidth represented in the IF data packet (in Hz).
- *center_frequency (64-bit integer): The center frequency of the IF data (in Hz).

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaSetSCPI:

Send a SCPI command to the RTSA. The list of all the SCPI commands as well as their definition can be found in the RTSA Programmer's Guide.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- *command(char array): The command to be sent to the RTSA

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaGetSCPI:

Send a SCPI command to the RTSA and wait for a response for the sent command. The list of all the SCPI commands as well as their definition can be found in the RTSA Programmer's Guide.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- *command(char array): The command to be sent to the RTSA.
- *query_response(char array): The command returned from the RTSA.

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaGetErrorMessage:

Retrieve an error message based on an error code. Note this is the same error code returned from the other API functions

Input Variables:

- *error_code(16-bit integer): The error code.

Return:

- An error message of (char array) containing information regarding the given error code.

wsaGetSweepSize:

Retrieve the array size required to store incoming sweep data. This function must be called before calling *wsaCaptureSpectrum* to configure the RTSA, and to allocate memory for the spectral data to be stored.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- fstart (unsigned 64-bit integer): The start frequency of the sweep (Hz).

- fstop (unsigned 64-bit integer): The stop frequency of the sweep (Hz).
- rbw (unsigned 32-bit integer): The RBW of the captured sweep (Hz).
- mode (char array): The RFE mode of the sweep capture (SH or SHN).
- attenuator (32-bit integer): Attenuation setting (0 or 1) of the 20 dB attenuator (only available on the RTSA-408 models).
- *sweep_size (unsigned 32-bit integer): The size of the array to be used to store the spectra data.

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaCaptureSpectrum:

Capture spectral data. The *wsaGetSweepSize* function must be called before calling this function to determine the array size of spectral_data.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- fstart (unsigned 64-bit integer): The start frequency of the sweep (Hz).
- fstop (unsigned 64-bit integer): The stop frequency of the sweep (Hz).
- rbw (unsigned 32-bit integer): The RBW of the captured sweep (Hz).
- mode (char array): The RFE mode of the sweep capture (SH or SHN).
- attenuator (32-bit integer): Attenuation setting (0 or 1) of the 20 dB attenuator (only available on the RTSA-408 models).
- *spectral_data(float): The spectral data of the given configuration

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaGetFFTSize:

Get the required array size to store the spectral data.

Input Variables:

- samples_per_packet (32-bit integer): The RTSA Session.
- stream_id (unsigned 32-bit integer): The stream id of the IF packet, which indicates what kind of data processing is required.
- *array_size (32-bit integer): The required array size to produce the buffer for the FFT calculation.

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaComputeFFT:

Compute the FFT, calculate the power spectrum, and apply a calibration factor to a time domain data. The *wsaGetFFTSize* function must be called before calling this function to pre-allocate memory for the power spectrum

Input Variables:

- samples_per_packet (32-bit integer): The RTSA Session.
- fft_size (32-bit integer): The size of the FFT buffer.
- stream_id (unsigned 32-bit integer): The stream id of the IF packet, which indicates what kind of data processing is required.
- reference_level (16-bit integer): The reference level containing the calibration value
- spectral_inversion (unsigned 8-bit integer):A flag which indicates whether the IF data contains any spectral inversion
- *i16_data (16-bit data buffer): A data buffer to store the 16-bit I-data. The value in this buffer will depend on the RFE mode of the RTSA.
- *q16_data (16-bit data buffer): A data buffer to store the 16-bit Q-data. The value in this buffer will depend on the RFE mode of the RTSA.
- *i32_data (32-bit data buffer): A data buffer to store the 32-bit I-data. This buffer will only contain data if the RTSA is set to 'HDR' RFE mode.
- *fft_buffer (float array): A buffer to store the produced FFT data. Note the size of this buffer is determined by the fft_size retrieved from the wsaGetFFTSize function.

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaPeakFind:

Read data within the specified parameters, compute the power spectrum and find the power level as well as the frequency of the peak within the power spectral data.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- fstart (unsigned 64-bit integer): The start frequency of the sweep (Hz).
- fstop (unsigned 64-bit integer): The stop frequency of the sweep (Hz).
- rbw (unsigned 32-bit integer): The RBW of the captured sweep (Hz).
- mode (char array): The RFE mode of the sweep capture (SH or SHN).
- ref_offset (float): A value used to apply any additional gain or loss such as cable lose (dB)
- attenuator (32-bit integer): Attenuation setting (0 or 1) of the 20 dB attenuator (only available on the RTSA-408 models).
- *peak_freq (unsigned 64-bit integer): The frequency of the peak power (Hz)
- *peak_power (float): The power level of the peak power (dBm)

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaChannelPower:

Read data within the specified parameters, compute the power spectrum and calculate the total integrated power within the frequency range of the channel bandwidth defined by the start and stop frequencies.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- fstart (unsigned 64-bit integer): The start frequency of the sweep (Hz).
- fstop (unsigned 64-bit integer): The stop frequency of the sweep (Hz).
- rbw (unsigned 32-bit integer): The RBW of the captured sweep (Hz).
- mode (char array): The RFE mode of the sweep capture (SH or SHN).
- ref_offset (float): A value used to apply any additional gain or loss such as cable lose (dB)
- attenuator (32-bit integer): Attenuation setting (0 or 1) of the 20 dB attenuator (only available on the RTSA-408 models).
- *channel_power (float): The channel power within the specified parameters (dBm)

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

wsaOccupiedBandwidth:

Compute the occupied bandwidth within the specified parameters. The occupied bandwidth is defined as the bandwidth containing a certain percentage of the total integrated power of the captured spectrum, centred on the assigned channel frequency. The percentage is user defined via the *occupied_percentage* parameter and a value of 92.5% is recommended.

Input Variables:

- *wsa_handle (64-bit integer): The RTSA Session.
- fstart (unsigned 64-bit integer): The start frequency of the sweep (Hz).
- fstop (unsigned 64-bit integer): The stop frequency of the sweep (Hz).
- rbw (unsigned 32-bit integer): The RBW of the captured sweep (Hz).
- occupied_percentage (float): Percentage of the total channel power which defines the occupied bandwidth
- mode (char array): The RFE mode of the sweep capture (SH or SHN).
- attenuator (32-bit integer): Attenuation setting (0 or 1) of the 20 dB attenuator (only available on the RTSA-408 models).
- *occupied_bandwidth (unsigned 64-bit integer): The bandwidth occupied by the signal of interest.

Return:

- A 16-bit integer error code, where negative code values indicate error occurrence. Use the wsaGetErrorMessage function to retrieve the message corresponding to the error code.

RTSA C++ API Example

The following is a subset of the files within the "\APIs\C++\Examples" directory of the RTSA Software and Firmware Release Package.

- calculateChannelPower.cpp: Connect, configure, and measure the channel power with specified parameters.
- calculateOccupiedBandwidth.cpp: Connect, configure, and measure the occupied bandwidth with specified parameters.
- captureSpectrum.cpp: Connect, configure, and capture spectral data.
- peakFind.cpp: Connect, configure, and measure the peak power and the frequency of the peak power.

Contact Us

ThinkRF Support website provides online documents for resolving technical issues with ThinkRF products at http://www.thinkrf.com/resources.

For all customers who hold a valid end-user license, ThinkRF provides technical assistance 9 AM to 5 PM Eastern Time, Monday to Friday. Contact us at http://www.thinkrf.com/support/, sales@thinkrf.com or by calling **+1.613.369.5104**.