TeraFAST software. C API reference

Terasense Group, Inc
2033 Gateway Place, Suite 500 San Jose, CA 95110, USA

January 18, 2024

Contents

1 Directory content

dlls\	
libterafast.dll	DLL files for the TeraView library and driver support software. Any software will depend on these DLLs.
okFrontPanel.dll	
driver\	
FrontPanelUSB-DriverOnly-4.4.0.exe	Device driver. Needs to be installed on the PC which would work with the device.
inc\	
libterafast.h	Header file for the TeraView library.
lib\	
libterafast.lib	Import library file for the TeraView library.
sample\	
sample.c	Source file for the simplest sample program.
sample.vcxproj	Visual Studio project for the simplest sample program.
Terasense Samples.sln	Visual Studio solution for the sample projects.

2 Introduction

The TeraView library provides C interface for developing programs for Terasense imaging devices models TeraFAST, and their custom modifications. It is compiled for Win32 platform and should work for Windows 7 and above.

In order to use library you need to include header file found in inc\ folder, link to import library file found in lib\ folder and put *both* DLL files found in dll\ folder within the search path (for example, in the same folder as your executable). You also need to install device driver found in driver\ folder (if you use Terasense Software on this computer, the driver is already installed).

The operation is started by initialization of the device (tfInit()). During this process a connection with the device is established, configuration data is read, and parameter variables are set. Next data acquisition is started (tfStart()). This step creates internal thread that constantly reads data from the device, process them, and stores result in internal buffer. Read operation tfRead() reads data from the internal buffer into preallocated buffer provided by you (use (TF_DATA) malloc(dataLength*sizeof(TF_ELEMENT)). It is guaranteed than the same data are never read twice and if there is no new data yet, the function blocks until they are available.

The data are read by frames with the width *frameSize*, which is determined by the parameters of your device and the length *frameLength*, which can be set using tfSetFrameLength() function (before the acquisition commences). The data are stored in the buffer line by line, earlier lines first. The rate of the line aquisition can be set by tfSetPeriod() or tfSetRate(), the latter is provided for convinience. Note that this is line acquisition rate, the frame rate would be *frameLength* times slower.

Data processing includes two steps: substracting backgound data and normalizing data. At the initialization device uses factory calibration (background and normalization), but you can record a new one using tfRecordBackground() and tfRecordNormalization() functions or load previously saved one using tfLoadConfig(). The normalization always contains two sets of data — default and recorded. Default never changes, tfRecordNormalization() replaces recorded set; if you've never used this function, the recorded data coincide with the default.

To free resources, call tfStop() when you do not need data, and tfClose() to free the device.

3 Reference

3.1 Types

typedef signed __int16 TF_ELEMENT
Type for a single data point.

typedef TF_ELEMENT * TF_DATA
Pointer to a buffer containing data points of a frame.

typedef int TF_RES
Type for return codes for functions in the library.

typedef int (*TF_ticker_t)(double completion)
Type for a ticker callback function being used by tfRecordBackground() and tfRecordNormalization(). Completion parameter varies from 0 (just started) to 1 (finished). The function should return FALSE by default and TRUE to request abort of the operation.

3.2 Variables

Variables are undefined before the initialization (see tfInit()). All variables are read-only, attempts to assign a new value will lead to crash!

int frameSize Dimension of the sensor.

int frameLength Length of a frame.

int dataLength Total number of data points in a frame ($dataLength = frameSize * frameLength$).

char deviceIDstring[32] Identification string for the device.

3.3 Constants

TF_MAX_VALUE 32767
Maximum value for a data point.

TF_DEFAULT_PERIOD 200
Default value for the time between data point (in microseconds).

TF_DEFAULT_FRAMELENGTH 64
Default value for the length of a frame.

TF_MAX_FRAMELENGTH 4096
Maximal value for the length of a frame.

TF_DEFAULT_THRESHOLD 10
Default value for the threshold.

3.4 Return codes

Name	Value	Description
TF_OK	0	Success.
TF_FAILED	-9999	General failure.
TF_CANCELLED	-9998	Cancelled by user.
TF_ERR_WRONG_PARAMETERS	-9997	Invalid parameters (out of range, too long or too short, etc.).
TF_NOTRUNNING	-9996	Data acquisition have not been started. (Use tfStart()).
TF_ERR_SIZE_MISMATCH	-9995	Imported data does not fit current configuration.
TF_OUTPUT_ERROR	-9994	File write operation failed.
TF_INPUT_ERROR	-9993	File read operation failed.
TF_NOTCONFIGURED	-9992	Initialization have not been performed or device have been closed. (Use tfInit()).
TF_ERR_RUNNING	-9991	Requested operation cannot be performed while data acquisition is running. (Use tfStop()).
TF_PATHNOTFOUND	-9990	File opening error - file or path cannot be found or permissions prevent it fom being opened or file is in use.
TF_TIMEOUT	-9989	The requested operation have not finished before timeout.

3.5 Functions

Main:

- **tfInit**(void)
- **tfClose**(void)
- **tfStart**(void)
- **tfStop**(void)
- **tfIsRunning**(void)
- **tfRead**(TF_DATA buffer)
- **tfRead_T**(TF_DATA buffer, int timeout)
- **tfReadRaw**(TF_DATA buffer)
- **tfReadRaw_T**(TF_DATA buffer, int timeout)

Acquisition settings:

- **SetFrameLength**(int length)
- **GetFrameLength**(void)

- **tfSetPeriod**(int period)
- **tfSetRate**(double rate)
- **tfGetPeriod**(void)
- **tfGetRate**(void)
- **tfGetPeriodRange**(int * min, int * max)
- **tfSetDifference**(int on)

Synchronization settings:

- **tfSetSyncOut**(int on)
- **tfGetSyncOut**(void)
- **tfSetExternalSync**(int on)
- **tfGetExternalSync**(void)
- **tfSetEdge**(int rise)
- **tfGetEdge**(void)
- **tfSetSyncDivider**(int div)
- **tfGetSyncDivider**(void)
- **tfSetSyncHoldoff**(int holdoff)
- **tfGetSyncHoldoff**(void)

Background and normalization:

- **tfRecordBackground**(TF_ticker_t callback)
- **tfRecordNormalization**(TF_ticker_t callback)
- **tfSelectNorm**(int i)
- **tfSaveConfig**(FILE *stream)
- **tfSaveConfigAs**(const char *filename)
- **tfLoadConfig**(FILE *stream)
- **tfLoadConfigFrom**(const char *filename)
- **tfSetThreshold**(double threshold)

3.5.1 Main

TF_RES tfInit(void);
Initialize the device. This function should be called prior to any operation with the device, which have to be connected to the computer. It performs initialization and sets variables *frameSize, frameLength, dataLength*, and *deviceIDstring*.

TF_RES tfClose(void);
Close the device. Device can be initialized again by calling tfInit().

TF_RES tfStart(void);
Starts data acquisition and processing. The acquisition and processing are performed continuously in separate threads and results are stored in internal cyclic buffer.

TF_RES tfStop(void);
Stops data acquisition and processing, joining corresponding threads. It can take as long as one frame period at current exposure to return.

int tfIsRunning(void);
Checks whether the acquisition is running. Returns 0 (false) or -1 (true).

TF_RES tfRead(TF_DATA buffer);
Reads processed data into the buffer (the buffer of the size $dataLength * sizeof(TS\_ELEMENT)$ must be allocated beforehand!). Returned data are between 0 and **TF_MAX_VALUE**.
If there are no new data, the function will block until they become available. If acquisition have not been started, the function will return **TF_NOTRUNNING**.

TF_RES tfRead_T(TF_DATA buffer, int timeout);
The same as above but it will return after *timeout* milliseconds. If timeout occurs it returns **TF_TIMEOUT**. Use this function if working with the external synchronization.

TF_RES tfReadRaw(TF_DATA buffer);
Reads raw (unprocessed) data into the buffer (the buffer of the size $dataLength * sizeof(TS\_ELEMENT)$ must be allocated beforehand!). No background compensation and calibration is applied. Returned data are approximately between **TF_RAW_LIMIT**−**TF_MAX_VALUE** and **TF_RAW_LIMIT**.
If there are no new data, the function will block until they become available. If acquisition have not been started, the function will return **TF_NOTRUNNING**.

TF_RES tfReadRaw_T(TF_DATA buffer, int timeout);
The same as above but it will return after *timeout* milliseconds. If timeout occurs it returns **TF_TIMEOUT**. Use this function if working with the external synchronization.

3.5.2 Acquisition settings

TF_RES tfSetFrameLength(int length);
Sets the number of lines in a single frame. This function can only be used when acquisition is not running. After using this function you'll need to adjust the size of the buffer.

int tfGetFrameLength(void);
Reads the number of lines in the current frame.

TF_RES tfSetPeriod(int period);
Set the time period between lines in microseconds. Admissible range can be found using tfGetPeriodRange(). If the parameter is out of range an error is returned.

TF_RES tfSetRate(double rate);
Set the acquisition rate in lines per second (note that this is line acquisition rate, frame rate is frameLength times slower). This is a helper function for SetPeriod(), the actual rate will be a rounded value corresponding to the nearest integer period.

int tfGetPeriod(void);
Returns the time period between lines in microseconds.

double tfGetRate(void);
Returns the acquisition rate in lines per second.

TF_RES tfGetPeriodRange(int * min, int * max);
Provides admissible range for the period value.

TF_RES tfSetDifference(int on);
Turns on/off difference mode. Non-zero value of parameter will turn it on, zero will turn it off.

3.5.3 Synchronization settings

TF_RES tfSetSyncOut(int on);
Turns sync output signal on or off.

int tfGetSyncOut(void);
Returns current status of the sync output signal.

TF_RES tfSetExternalSync(int on);
Turns external sync mode on or off. tfRead() and tfReadRaw() will block if you call this function to turn external sync mode on but no sync signal is supplied to the corresponding input. Use timeout versions to avoid it tfRead_T() and tfReadRaw_T().

int tfGetExternalSync(void);
Returns current status of the external sync mode.

TF_RES tfSetEdge(int rise);
Sets trigger edge for external sync signal. Use 0 for the falling edge, 1 for the rising edge.

int tfGetEdge(void);
Returns current status of the trigger edge.

TF_RES tfSetSyncDivider(int div);
Sets divider for the external sync signal frequency, $1 \leq div \leq 32768$.

int tfGetSyncDivider(void);
Returns current value of the divider.

TF_RES tfSetSyncHoldoff(int holdoff);
Sets holdoff parameter for the external sync signal, $0 \leq holdoff \leq 15$. It corresponsd to minimum $2^h oldoff$ microseconds delay between active edges.

int tfGetSyncHoldoff(void);
Returns current value of the holdoff parameter.

3.5.4 Background and normalization

TF_RES tfRecordBackground(TF_ticker_t callback);
Records background compensation data. The radiation source should be off. The data are recording can take some time, depending on acquisition rate. The parameter is a pointer to callback function which is called periodically to indicate progress. Return value of this function can be used to cancel the process. Set the parameter to **NULL** to prevent callbacks.
The recorded data are applied automatically and replace default during the session, to use them in subsequent sessions you need to save and then load config.

TF_RES tfRecordNormalization(TF_ticker_t callback);
Records normalization data. The radiation source should be on and it is advisable to record background data beforehand. The data are recording can take some time, depending on acquisition rate. The parameter is a pointer to callback function which is called periodically to indicate progress. Return value of this function can be used to cancel the process. Set the parameter to **NULL** to prevent callbacks.
The recorded data are NOT applied automatically. In order to apply them, use tfSelectNorm(1). To use the recorded data in subsequent sessions you need to save and then load config.

TF_RES tfSelectNorm(int i);
Selects nomaliztion (0 - default, 1 - recorded). In the default configuration recorded normalization coincides with the default.

TF_RES tfSaveConfig(FILE *stream);
Saves calibration data (i.e. background and normalization data) to file. The parameter should be a binary stream opened for writing.

TF_RES tfSaveConfigAs(const char *filename);
TF_RES tfSaveConfigAs_w(const _wchar_t *filename);
Helper functions that save configuration to the file indicated by filename. If the file already exists, it is overwritten silently; if it cannot be open for any reason, functions return **TF_PATHNOTFOUND**. The first function takes ANSI string as argument, while the second takes wide-character string.

TF_RES tfLoadConfig(FILE *stream);
Reads calibration data (i.e. background and normalization) from file. The parameter should be a binary stream opened for reading. The data read replace default during the session.

TF_RES tfLoadConfigFrom(const char *filename);
TF_RES tfLoadConfigFrom_w(const _wchar_t *filename);
Helper functions that load configuration from the file indicated by filename. If the file cannot be open for any reason, functions return **TF_PATHNOTFOUND**. The first function takes ANSI string as argument, while the second takes wide-character string.

TF_RES tfSetThreshold(double threshold);

During normalization recording process SNR for the pixels is calculated. Pixels with SNR below a certain level are marked as non-performing and their data are replaces by zeros when using the recorded normalization data. This function sets the corresponding threshold.
If argument is below 1, the **TF_DEFAULT_THRESHOLD** is used.

4 Samples

For the expample of a simplest program see included 'sample' project.