TERIDIAN SEMICONDUCTOR CORP.

Simplifying System Integration™

73M1822/73M1922
Modem API User Guide

December 23, 2009
Rev. 1.0
UG_1x22_055

Teridian Semiconductor Corp., 6440 Oak Canyon, Suite 100, Irvine, CA 92618
TEL (714) 508-8800, FAX (714) 508-8877, http://www.teridian.com

Table of Contents

Figures

Tables

1 Introduction

This document describes the application programming interface (API) of the Teridian 73M1x22 Modem API. The API is defined as a simplified functional interface that allows an application program written in "C" language to communicate with the driver without going through complicate and cumbersome Linux® driver IOCTL. In addition, the API subsystem also encapsulates the asynchronous event notification and provides a much simpler mechanism to the user application via a callback function.

The API subsystem resides between the user application and the driver layer as shown in Figure 1.

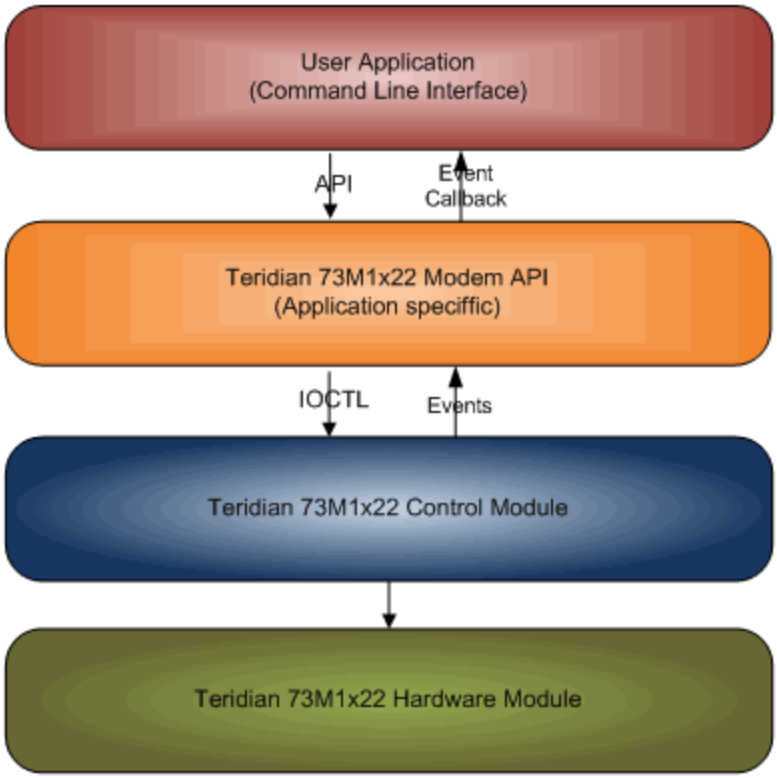

Figure 1: Driver API

1.1 Overview

The Modem API is a user-friendly programming conduit used to access and manage the modem device/channel via the Reference Driver. This section describes the programming flow and how the application program can be implemented using the Modem APIs.

First, the concept of device versus channel, the Teridian 73M1x22 Control module supports multiple modem channels. The application layer communicates with the driver using this channel descriptor. However, the driver reports asynchronous channel events through the device descriptor. At this point, the only use for the device descriptor is for conveying event notification and for retrieving events from the driver.

The application program must first initialize the Modem API by calling `M1X22_MdmApiInit()`. This initializes internal data structures and prepares the API subsystem for subsequent call to other APIs.

After the Modem API is initialized, the application program opens the device and channel descriptors to access and manage the device and channel(s) by calling `M1X22_OpenDevice()` and `M1X22_OpenChannel()` respectively. If more than one channel exists in the hardware platform, each channel is opened separately by calling `M1X22_OpenChannel()`.

To simplify the usage of the driver, the API encapsulates all relevant data components required to operate the device and the channels into a data structure whose pointer is referred to as a “handle”. This handle is returned from a successful call to open device/channel APIs and is used throughout the life of the device and channels.

Upon exiting the application program it is recommended that all device and channels descriptors be closed, using `M1X22_CloseDevice()` and `M1X22_CloseChannel()`, and the API subsystem be shutdown using `M1X22_MdmApiRelease()` to ensure that the system resources are free up when the application program terminates.

As mentioned earlier, the asynchronous events from each channel are reported through the opened device descriptor. The event message contains the channel ID from which the event occurred on. The application program must distinguish the event for each channel and act accordingly. The way this event is conveyed to the application program is via the callback function provided as one of the parameter passed in to the `M1X22_OpenDevice()`. The callback function serves as a “hook” for the API subsystem and notifies the application whenever an event is generated.

Section 5 provides a sample application using the modem API.

1.2 Conventions Used in this Guide

This document uses the following conventions:

- Software code, IOCTL names and data types are presented in Courier font.
- A table with a blue header is a summary table. A table with a gray header is a detail table.

1.3 Acronyms

API – Application Programming Interface
APOH – Another Phone Off Hook
BSP – Board Support Package
DAA – Data Access Arrangement
IET – Interval Event Table
IOCTL – I/O Control
NOPOH – No Phone Off Hook
POH – Phone Off Hook

2 API Descriptions

This section contains the detail description of each API. Table 1 provides a summary of the APIs.

Table 1: API Overview

IOCTL Name	Description
M1X22_MdmApiInit	API subsystem initialization.
M1X22_MdmApiRelease	API subsystem release or termination.
M1X22_OpenDevice	Open device of event reporting.
M1X22_CloseDevice	Close an opened device.
M1X22_OpenChannel	Open channel for operation.
M1X22_CloseChannel	Close an opened channel.
M1X22_InitChannel	Initialize channel for specific region.
M1X22_HookSwitch	Perform hook switching – on/off-hook.
M1X22_PulseDial	Pulse dialing.
M1X22_HWRegisterRead	Read 1x22 register.
M1X22_HWRegisterReadAll	Read all registers.
M1X22_HWRegisterWrite	Write to 1x22 register.
M1X22_SetDebugTrace	Set debug trace mask.
M1X22_SetPhoneVolume	Set speaker phone and microphone volume.
M1X22_CallProgressMonitor	Monitor activities on the line.
M1X22_GetFileDescriptor	Get file descriptor from a handle.
M1X22_GetCurrentIET	Read current IET entry.
M1X22_UpdateCurrentIET	Update current IET entry.
M1X22_ClearCurrentIET	Delete the current IET table entry.
M1X22_GetVoltageIET	Read the voltage IET table entry.
M1X22_UpdateVoltageIET	Update voltage IET entry.
M1X22_ClearVoltageIET	Delete the voltage IET table entry.
M1X22_StartMeasureCurrent	Start current measurement.
M1X22_StartMeasureVoltage	Start voltage measurement.
M1X22_StopMeasureCurrent	Stop current measurement.
M1X22_StopMeasureVoltage	Stop voltage measurement.
M1X22_SetRingCadence	Set Ring Cadence filtering criteria.

2.1 M1X22_MdmApiInit

Description

This API allocates and initializes required internal data structures; make necessary preparation before other API members can be invoked. The `M1X22_MdmApiInit()` must be called once during application program's initialization, or prior to accessing other API members. Upon exiting of the application program a reversed operation must be performed to release resources using `M1X22_MdmApiRelease`.

Prototype

```
M1X22_RET M1X22_MdmApiInit (void);
```

Parameters

Data Type	Name	Description
`void`	`void`	No parameter is required.

Return Values

Data Type	Description
`M1X22_RET`	M1X22_SUCCESS – Successful API initialization. M1X22_FAILED – Failed.

2.2 M1X22_MdmApiRelease

Description

This API de-allocates the resources acquired during initialization – `M1X22_MdmApiInit`. It must be called prior to application program termination, or when access to API is no longer needed. Any channels or devices remaining open when this API is called will be closed.

Prototype

```
M1X22_RET M1X22_MdmApiRelease (void);
```

Parameters

Data Type	Name	Description
`void`	`void`	No parameter is required.

Return Values

Data Type	Description
`M1X22_RET`	M1X22_SUCCESS – Successful API termination. M1X22_FAILED – Unsuccessful.

2.3 M1X22_OpenDevice

Description

This API performs the low level opening of the device descriptor. By default, the device descriptor name is known as "/dev/tsc_1x22_ctl_device"; however, since this is configurable during driver installation, a different device descriptor name may be used. Refer to driver configuration and installation more detail. Calling to this open device API is mandatory before any channel descriptor can be opened, and it must be called after the API Init function.

In addition to opening the device for accessing this API also requires a callback function. This function is called by the API subsystem to convey the FOX events to the application program. Six parameters are returned when this callback function is invoked:

`eventId`: This parameter is the event identification of type `M1X22_EVENT_ID`.
`chanId`: This parameter contains the channel ID from which the event is generated.
`evtData1:` are reserved for special purpose in event handling.
`evtData2:`
`evtData3:`
`evtData4:`

A successful open device returns a non-zero device handle that can be used to access other device related APIs.

Prototype

```
M1X22_HANDLE M1X22_OpenDevice (char *devName, void *callback);
```

Parameters

Data Type	Name	Description
`char *`	`devName`	Device descriptor name (default: /dev/ter10).
`void`	`(*eventCallback)( M1X22_EVENT_ID eventId, unsigned int chanId, unsigned int evtData1, unsigned int evtData2, unsigned int evtData3, unsigned int evtData4)`	Event status callback function. The parameters returned by this callback function are event ID, channel ID and 4 event data that are specific to the event ID. Most of these event data are not applicable except those listed in the "EventCallback Parameters" table below.

EventCallback Parameters

Event Name	Parameters
`M1X22_EVENT_RING_START`	`evtData1` – Ring burst frequency (in Hz).
`M1X22_EVENT_RING_END`	`evtData1` – Ring burst frequency (in Hz). `evtData2` – Ring burst duration (in milliseconds).
`M1X22_EVENT_LINE_STATE`	`evtData1` – Channel Hook state (0-on-hook, 1-off-hook). `evtData2` – 0-line current, 1- voltage. `evtData3` – IET index. `evtData4` – Application defined event.

Return Values

Data Type	Description
`M1X22_HANDLE`	Non-ZERO value if the device is opened successfully. ZERO if the device fails to open.

2.4 M1X22_CloseDevice

Description

This API performs the low level closing of the device handle. It should be called before exiting application program to release all resource associated with the open device. Closing this device will automatically force the closing of any opened channel handles.

Prototype

```
M1X22_RET M1X22_CloseDevice (M1X22_HANDLE devHandle);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	devHandle	Device handle obtained from open device.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully close the device handle. M1X22_FAILED – Failed to close the handle.

2.5 M1X22_OpenChannel

Description

This API performs the low level opening of the channel descriptor. By default, the channel descriptor is known as "/dev/tsc_1x22_ctl_channelxx", where XX varies from 1 to 16 representing channel 0 to 15 respectively. Calling to the open channel API is mandatory before any access to the channel can be performed, and it must be called after the open device as this API requires the device handle to be passed in as the second parameter. A successful open channel returns a non-zero channel handle that can be used to access other channel related APIs.

Prototype

```
M1X22_RET M1X22_OpenChannel (
      char *pChanName,
      M1X22_HANDLE devHandle,
      int chanNum);
```

Parameters

Data Type	Name	Description
char *	pChanName	Channel descriptor name. (Default: /dev/ter11...26).
M1X22_HANDLE	devHandle	Device handle obtained from open device.
int	chanNum	Channel number from 0 to 15.

Return Values

Data Type	Description
M1X22_HANDLE	0 – Failed to open channel. None-zero handle – Successful.

2.6 M1X22_CloseChannel

Description

This API performs the low level closing of the channel handle. It should be called before exiting application program, or when the channel is no longer needed, to release all resources associated with the open channel.

Prototype

```
M1X22_RET M1X22_CloseChannel (M1X22_HANDLE chanHandle);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	chanHandle	Channel handle obtained from open channel.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully close the channel. M1X22_FAILED – Failed to close the channel.

2.7 M1X22_InitChannel

Description

This API performs the channel initialization and prepares the channel to operate in conformance with specific region where the line is being deployed. Teridian device supports global-compliance parameter in countries listed in M1X22_COUNTRY_CODE.

Prototype

```
M1X22_RET M1X22_InitChannel (
      M1X22_HANDLE chanHandle,
      M1X22_COUNTRY_CODE cntryCode );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	chanHandle	Channel handle obtained from open channel.
M1X22_COUNTRY_CODE	cntryCode	Country code as listed in M1X22_COUNTRY_CODE.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully initialize the channel. M1X22_FAILED – Failed to initialize the channel.

2.8 M1X22_HookSwitch

Description

Perform on/off-hook on the channel.

Prototype

```
M1X22_RET M1X22_HookSwitch (
      M1X22_HANDLE channel,
      M1X22_HOOK_SWITCH hookSwitch );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle from open channel.
M1X22_HOOK_SWITCH	hookSwitch	Type of hook switch requested.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully performed hook switch. M1X22_FAILED – Failure.

2.9 M1X22_PulseDial

Description

Perform pulse dialing on the channel. This API will bring the channel off-hook then perform pulse dialing with the digit string provided. Once the dialing is completed the channel will remain off-hook indefinitely, or until it is manually brought back on-hook.

Up to 30 numeric digits is supported. Valid numeric digit is from ASCII 0 to 9 (i.e., 0x30 to 0x39).

Prototype

```
M1X22_RET M1X22_PulseDial (
      M1X22_HANDLE channel,
      char *dialSting );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle from open channel.
char	*dialString	Numeric dial string of up to 30 characters max.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully dialed. M1X22_FAILED – Failure.

2.10 M1X22_HWRegisterRead

Description

Perform H/W register reading from the 73M1x22 device that carries the channel. This API is not recommended for use in normal operation but only for diagnostic or testing purpose.

Prototype

```
M1X22_RET M1X22_HWRegisterRead (
      M1X22_HANDLE channel,
      M1X22_REG_TYPE register,
      char *pRetData );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle from open channel.
M1X22_REG_TYPE	register	1x22 register.
char	*pRetData	Pointer to a single byte of returned data.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully read the register. M1X22_FAILED – Failure.

2.11 M1X22_HWRegisterReadAll

Description

This API is similar to the M1X22_HWRegisterRead, but all 73M1x22 registers will be read. The register contents will be returned via the pRetData. It is the responsibility of the caller application to insure that this data pointer contains enough space for the return data. A 0x20 bytes buffer is required to store all returned register where the first byte corresponds to the first register (RG00) and so on.

Prototype

```
M1X22_RET M1X22_HWRegisterReadAll (
      M1X22_HANDLE channel,
      char *pRetData );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle from open channel.
char	*pRetData	Pointer to return data for all 0x20 (byte) registers.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully read all registers. M1X22_FAILED – Failure.

2.12 M1X22_HWRegisterWrite

Description

Perform H/W register writing to the 73M1x22 device that carries the channel. This API is not recommended for use in normal operation but only for diagnostic or testing purpose.

Prototype

```
M1X22_RET M1X22_HWRegisterWrite (
      M1X22_REG_TYPE register,
      char value );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle from open channel.
M1X22_REG_TYPE	register	1x22 Register.
char	value	Data value to be written.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully written the register. M1X22_FAILED – Failure.

2.13 M1X22_SetDebugTrace

Description

The M1X22 Reference Driver provides run-time debugging facility via tracing. The application program can turn on various trace masks to display debug message for trouble shooting purposes. Due to the potential amount of messages resulting from setting this trace mask at kernel level the use of this facility is not recommended under normal operation.

Note: the trace mask occupies a unique bit field, therefore, multiple masks can be “or” together.

Prototype

```
M1X22_RET M1X22_SetDebugTrace (
      M1X22_HANDLE deviceHandle,
      M1X22_DEBUG_TRACE_MASK debugTraceMask );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	deviceHandle	Device handle from open device.
M1X22_DEBUG_TRACE_MASK	debugTraceMask	Debug trace mask.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully set the trace mask. M1X22_FAILED – Failure.

2.14 M1X22_SetPhoneVolume

Description

Set the speaker phone and microphone volume. This API sets transmit and receive dB gain of the channel.

Prototype

```
M1X22_RET M1X22_SetPhoneVolume (
      M1X22_HANDLE chanHandle,
      int txVolumedB,
      int rxVolumedB );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	chanHandle	Channel handle obtains from open channel.
int	txVolumedB	Transmit direction in dB.
int	rxVolumedB	Receive direction in dB.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully set phone volume. M1X22_FAILED – Failure.

2.15 M1X22_CallProgressMonitor

Description

For the purpose of monitoring activities on the line, a Call Progress Monitor is provided in the 73M1x22. This audio output contains both transmit and receive data with configurable levels.

Prototype

```
M1X22_RET M1X22_CallProgressMonitor (
      M1X22_HANDLE      chanHandle,
      M1X22_CM_VOLTAGE_REF voltRef,
      M1X22_CM_GAIN     txGain,
      M1X22_CM_GAIN     rxGain);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	chanHandle	Channel handle obtains from open channel.
M1X22_CM_VOLTAGE_REF	voltRef	Voltage Reference selection
M1X22_CM_GAIN	txGain	Transmit path gain setting
M1X22_CM_GAIN	rxGain	Receive path gain setting

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successful. M1X22_FAILED – Failure.

2.16 M1X22_GetFileDescriptor

Description

To provide full flexibility to the driver access, this API can be called by an application program to obtain the file descriptor for a given device or channel handle. With this returned file descriptor, the user has direct access to the driver via its native IOCTL as described in the *73M1822/73M1922 Control Module User Guide*.

Caution: Accessing directly to the driver via IOCTL in mix mode with API is generally not recommended. However, if this becomes necessary it is strongly advised that the user carefully review the API implementation and be aware of any potential conflicts that may exist as a result of this change.

Prototype

```
int M1X22_GetFileDescriptor (M1X22_HANDLE pHandle);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	pHandle	Device or channel handle obtains from open API.

Return Values

Data Type	Description
int	Device or channel file descriptor. A NULL value indicates invalid, or not opened file descriptor.

2.17 M1X22_GetCurrentIET

Description

Read the line current IET table entry.

Prototype

```
int M1X22_GetCurrentIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex,
      M1X22_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1X22_IET_t	*pIET	Pointer to an output structure M1X22_IET_t.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully read the IET entry. M1X22_FAILED – Failure.

2.18 M1X22_UpdateCurrentIET

Description

Perform the line current IET table update.

Prototype

```
int M1X22_UpdateCurrentIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex,
      M1X22_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1X22_IET_t	*pIET	Pointer to an input structure M1X22_IET_t.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully update the IET entry. M1X22_FAILED – Failure.

2.19 M1X22_ClearCurrentIET

Description

Delete the line current IET table entry.

Prototype

```
int M1X22_ClearCurrentIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully clear the IET entry. M1X22_FAILED – Failure.

2.20 M1X22_GetVoltageIET

Description

Read the line voltage IET table entry.

Prototype

```
int M1X22_GetVoltageIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex,
      M1X22_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1X22_IET_t	*pIET	Pointer to an output structure M1X22_IET_t.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully read the IET entry. M1X22_FAILED – Failure.

2.21 M1X22_UpdateVoltageIET

Description

Perform the line voltage IET table update.

Prototype

```
int M1X22_UpdateVoltageIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex,
      M1X22_IET_t  *pIET );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).
M1X22_IET_t	*pIET	Pointer to an input structure M1X22_IET_t.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully update the IET entry. M1X22_FAILED – Failure.

2.22 M1X22_ClearVoltageIET

Description

Delete the line voltage IET table entry.

Prototype

```
int M1X22_ClearVoltageIET (
      M1X22_HANDLE channel,
      unsigned int ietIndex);
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	ietIndex	IET entry index – row number (0 – 9).

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully clear the IET entry. M1X22_FAILED – Failure.

2.23 M1X22_StartMeasureCurrent

Description

Start the line current measurement process. This API starts measuring the line current of the specified channel at a sampling interval time of “sampleTime” with averaging count of “averageSampleCount”.

Prototype

```
int M1X22_StartMeasureCurrent (
      M1X22_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully started. M1X22_FAILED – Failure.

2.24 M1X22_StartMeasureVoltage

Description

Start the line voltage measurement process.

Prototype

```
int M1X22_StartMeasureVoltage (
      M1X22_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully started. M1X22_FAILED –Failure.

2.25 M1X22_StopMeasureCurrent

Description

Stop the line current measurement.

Prototype

```
int M1X22_StopMeasureCurrent (
      M1X22_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully stopped. M1X22_FAILED – Failure.

2.26 M1X22_StopMeasureVoltage

Description

Stop the line voltage measurement.

Prototype

```
int M1X22_StopMeasureVoltage (
      M1X22_HANDLE channel,
      unsigned int sampleTime,
      unsigned int averageSampleCount );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	sampleTime	Time interval between two consecutive samples.
unsigned int	averageSampleCount	Sample count for average calculation.

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successfully stopped. M1X22_FAILED – Failure.

2.27 M1X22_SetRingCadence

Description

Set the ring cadence parameter for ring qualification process. The driver can be programmed to filter and qualify incoming ring by discarding spurious signals that can be caused by line noises or distortions. When set with these cadence parameters the driver will evaluate each incoming ring cycle based on these parameters and emits M1X22_EVENT_QUALIFIED_RING event only when the ring cadence matches at the specified number of cycles.

The ring cadence parameter consists of the frequency, cycle duration, period-1 “on”, period-1 “off”, period-2 “on”, period-2 “off”, and number of cycles. This last parameter (number of cycle) is the number of ring cycles matched before the ring is qualified. The sum of the period-1 “on”, period-1 “off”, period-2 “on” and period-2 “off” must be equal to the cycle duration parameter.

Remark

The system is defaulted to ring cadence screening disabled upon startup. Enabling or disabling the filter using this API will not affect the behavior of the ring detection events (M1X22_EVENT_RING_START and M1X22_EVENT_RING_END). The application may chose to ignore these two events if the qualified ring event is preferred. To turn off this filter the same API can be used with the frequency or cycle duration parameter equal ZERO.

Prototype

```
int M1X22_SetRingCadence (
      M1X22_HANDLE channel,
      unsigned int frequency,
      unsigned int cycleDuration,
      unsigned int periodOneOn,
      unsigned int periodOneOff,
      unsigned int periodTwoOn,
      unsigned int periodTwoOff,
      unsigned int numOfCycles );
```

Parameters

Data Type	Name	Description
M1X22_HANDLE	channel	Channel handle obtained from open channel.
unsigned int	frequency	Ring frequency to be detected.
unsigned int	cycleDuration	Duration of the cadence cycle (in milliseconds).
unsigned int	periodOneOn	Period-1 “On” duration (in milliseconds).
unsigned int	periodOneOff	Period-1 “Off” duration (in milliseconds).
unsigned int	periodTwoOn	Period-2 “On” duration (in milliseconds).
unsigned int	periodTwoOff	Period-2 “Off” duration (in milliseconds).
unsigned int	numOfCycles	Detect at the end of this cycle number (1 – 5).

Return Values

Data Type	Description
M1X22_RET	M1X22_SUCCESS – Successful. M1X22_FAILED – Failure.

3 Structure Reference

This section contains the detail description of the data structure used in this API subsystem. Table 2 contains the summary of the reference structure.

Table 2: Structure Overview

Name	Description
M1X22_HANDLE	Device or Channel handle.
M1X22_CHAN_INIT	Channel Init structure.

3.1 M1X22_HANDLE

Description

The device or channel handle is a pointer to a device or channel control data structure where information pertaining to device or channel is being kept. This control data structure is created when the device or channel is opened and is used by the API internal subsystem to indentify the device or channel until it is closed.

Prototype

```
/*
*****************************************************************************
** Device/Channel Control block
*****************************************************************************/
typedef struct {
    int     chanNum;
    int     fd;
    void    (*eventCallback)(M1X22_EVENT_ID,
                             unsigned int,
                             unsigned int,
                             unsigned int,
                             unsigned int,
                             unsigned int);
    int     initialized;
}
m1X22_DevChanHandle;

/* Device or channel handle */
typedef m1X22_DevChanHandle *M1X22_HANDLE;
```

Parameters

Data Type	Name	Description
int	chanNum	Device or channel number.
int	fd	Device or channel corresponding file descriptor.
void	(*eventCallback)()	An event callback function pointer, used only for device.
int	initialized	Initialization flag.

3.2 M1X22_CHAN_INIT

Description

Channel initialization structure. This structure contains mainly the country code. The `M1X22_InitChannel` API passes the pointer to this data structure to configure the channel to operate in a specific region. Refer to M1X22_COUNTRY_CODE for the list of countries supported.

Prototype

```
/*
*****************************************************************************
** Channel Init Structure
*****************************************************************************/
typedef struct {
    unsigned char      nMode;
    M1X22_COUNTRY_CODE nCountry;
    void               *pProc;
}
m1X22_ChanInit;
typedef m1X22_ChanInit M1X22_CHAN_INIT;
```

Parameters

Data Type	Name	Description
`char`	`nMode`	N/A
`M1X22_COUNTRY_CODE`	`nCountry`	Defined country codes.
`void *`	`pProc`	N/A

4 Enumerator Reference

This section contains the detail description of the enumerated data reference used in this API subsystem. Table 3 contains the summary of the enumerated data reference.

Table 3: Enumerator Overview

Name	Description
M1X22_RET	Return status code from API.
M1X22_EVENT_ID	Event identification.
M1X22_COUNTRY_CODE	List of country code supported.
M1X22_HOOK_SWITCH	On and off hook.
M1X22_REG_TYPE	List of 1x22 device H/W register.
M1X22_DEBUG_TRACE_MASK	List of debug trace mask.

4.1 M1X22_RET

Description

API function return code. All APIs, expect the open device and open channel, return with M1X22_RET code. For additional error information when an API returns M1X22_FAILED the application program can issue a get last error – M1X22_GetLastError.

Prototype

```
/*
*****************************************************************************
** Function return code
*****************************************************************************/
typedef enum {
    M1X22_SUCCESS  = 0,
    M1X22_FAILED   = (-1),
}
M1X22_RET;
```

Name	Value	Description
M1X22_SUCCESS	0	Successful.
M1X22_FAILED	-1	Failed.

4.2 M1X22_EVENT_ID

Description

List of event identification supported by the API subsystem.

Prototype

```
/*
**************************************************************************
** Event ID
**************************************************************************/
typedef enum {
    M1X22_EVENT_NOPOH         = M1X22_NOPOH_DETECT,
    M1X22_EVENT_APOH          = M1X22_APOH_DETECT,
    M1X22_EVENT_POLARITY_REV  = M1X22_POLARITY_CHG,
    M1X22_EVENT_BATT_DROP     = M1X22_BATTERY_DROPPED,
    M1X22_EVENT_BATT_FEED     = M1X22_BATTERY_FEEDED,
    M1X22_EVENT_RING_END      = M1X22_RING_DETECT_END,
    M1X22_EVENT_RING_START    = M1X22_RING_DETECT,
    M1X22_EVENT_SYNC_LOST     = M1X22_SYNC_LOSS_DETECT,
    M1X22_EVENT_ON_HOOK       = M1X22_ON_HOOK_DETECT,
    M1X22_EVENT_OVER_VOLTAGE  = M1X22_OV_DETECT,
    M1X22_EVENT_OVER_CURRENT  = M1X22_OI_DETECT,
    M1X22_EVENT_LINE_STATE    = M1X22_LINE_STATE,
    M1X22_EVENT_UNDER_VOLTAGE = M1X22_UV_DETECT,
    M1X22_EVENT_QUALIFIED_RING= M1X22_QUALIFIED_RING,
    M1X22_EVENT_DIAL_COMPLETE = M1X22_DIAL_COMPLETE,
    M1X22_EVENT_DIAL_ABORTED  = M1X22_DIAL_ABORTED,
    M1X22_EVENT_GPIO_INTERRUPT= M1X22_GPIO_INTERRUPT
}
M1X22_EVENT_ID;
```

Parameters

Name	Value	Description
M1X22_EVENT_GPIO_INTERRUPT	0x00800000	GPIO interrupt.
M1X22_EVENT_NOPOH	0x00400000	No other phone off-hook.
M1X22_EVENT_APOH	0x00200000	Another phone off-hook.
M1X22_EVENT_POLARITY_REV	0x00100000	Line polarity changed.
M1X22_EVENT_BATT_DROP	0x00080000	Battery – line is not feeded.
M1X22_EVENT_BATT_FEED	0x00040000	Battery – line is feeded.
M1X22_EVENT_RING_END	0x00020000	Line stopped ringing.
M1X22_EVENT_RING_START	0x00010000	Line starts ringing.
M1X22_EVENT_SYNC_LOST	0x00008000	Device lost synchronization.
M1X22_EVENT_ON_HOOK	0x00002000	Line goes on-hook.
M1X22_EVENT_OVER_VOLTAGE	0x00001000	Over voltage condition detected.
M1X22_EVENT_OVER_CURRENT	0x00000800	Over current condition detected.
M1X22_EVENT_UNDER_VOLTAGE	0x00000200	Under voltage condition detected.
M1X22_EVENT_LINE_STATE	0x00000100	Line state event – current/voltage.
M1X22_EVENT_QUALIFIED_RING	0x00000080	Qualified ring cadence detected.
M1X22_EVENT_DIAL_COMPLETE	0x00000040	Pulse Dial complete.
M1X22_EVENT_DIAL_ABORTED	0x00000020	Pulse Dial aborted.
M1X22_EVENT_SYNC_RECOVERED	0x00000010	Barrier sync recovered.

4.3 M1X22_COUNTRY_CODE

Description

List of country codes supported by the API subsystem.

Prototype

```
/*
**************************************************************************
**  Country code List - Internet Country Codes
**************************************************************************/
typedef enum {
    M1X22_CNTRY_CODE_AR          = 0,      /* "Argentina"    */
    M1X22_CNTRY_CODE_AU          = 1,      /* "Australia"    */
    M1X22_CNTRY_CODE_AT          = 2,      /* "Austria"      */
    M1X22_CNTRY_CODE_BH          = 3,      /* "Bahrain"      */
    M1X22_CNTRY_CODE_BE          = 4,      /* "Belgium"      */
    M1X22_CNTRY_CODE_BR          = 5,      /* "Brazil"       */
    M1X22_CNTRY_CODE_BG          = 6,      /* "Bulgaria"     */
    M1X22_CNTRY_CODE_CA          = 7,      /* "Canada"       */
    M1X22_CNTRY_CODE_CL          = 8,      /* "Chile"        */
    M1X22_CNTRY_CODE_C1          = 9,      /* "ChineData"    */
    M1X22_CNTRY_CODE_C2          = 10,     /* "ChinaVoice"   */
    M1X22_CNTRY_CODE_CO          = 11,     /* "Columbia"     */
    M1X22_CNTRY_CODE_HR          = 12,     /* "Croatia"      */
    M1X22_CNTRY_CODE_TB          = 13,     /* "TBR 21"       */
    M1X22_CNTRY_CODE_CY          = 14,     /* "Cyprus"       */
    M1X22_CNTRY_CODE_CZ          = 15,     /* "Czech Rep"    */
    M1X22_CNTRY_CODE_DK          = 16,     /* "Denmark"      */
    M1X22_CNTRY_CODE_EC          = 17,     /* "Equador"      */
    M1X22_CNTRY_CODE_EG          = 18,     /* "Egypt"        */
    M1X22_CNTRY_CODE_SV          = 19,     /* "El Salvador"  */
    M1X22_CNTRY_CODE_FI          = 20,     /* "Finland"      */
    M1X22_CNTRY_CODE_FR          = 21,     /* "France"       */
    M1X22_CNTRY_CODE_DE          = 22,     /* "Germany"      */
    M1X22_CNTRY_CODE_GR          = 23,     /* "Greece"       */
    M1X22_CNTRY_CODE_GU          = 24,     /* "Guam"         */
    M1X22_CNTRY_CODE_HK          = 25,     /* "Hong Kong"    */
    M1X22_CNTRY_CODE_HU          = 26,     /* "Hungary"      */
    M1X22_CNTRY_CODE_IS          = 27,     /* "Iceland"      */
    M1X22_CNTRY_CODE_IN          = 28,     /* "India"        */
    M1X22_CNTRY_CODE_ID          = 29,     /* "Indonesia"    */
    M1X22_CNTRY_CODE_IE          = 30,     /* "Ireland"      */
    M1X22_CNTRY_CODE_IL          = 31,     /* "Israel"       */
    M1X22_CNTRY_CODE_IT          = 32,     /* "Italy"        */
    M1X22_CNTRY_CODE_JP          = 33,     /* "Japan"        */
    M1X22_CNTRY_CODE_JO          = 34,     /* "Jordan"       */
    M1X22_CNTRY_CODE_KZ          = 35,     /* "Kazakhstan"   */
    M1X22_CNTRY_CODE_KW          = 36,     /* "Kuwait"       */
    M1X22_CNTRY_CODE_LV          = 37,     /* "Latvia"       */
    M1X22_CNTRY_CODE_LB          = 38,     /* "Lebanon"      */
    M1X22_CNTRY_CODE_LU          = 39,     /* "Luxembourg"   */
    M1X22_CNTRY_CODE_MO          = 40,     /* "Macao"        */
    M1X22_CNTRY_CODE_MY          = 41,     /* "Malaysia"     */
    M1X22_CNTRY_CODE_MT          = 42,     /* "Malta"        */
    M1X22_CNTRY_CODE_MX          = 43,     /* "Mexico"       */
    M1X22_CNTRY_CODE_MA          = 44,     /* "Morocco"      */
    M1X22_CNTRY_CODE_NL          = 45,     /* "Netherlands"  */
    M1X22_CNTRY_CODE_NZ          = 46,     /* "New Zealand"  */
```

```
    M1X22_CNTRY_CODE_NG           = 47,     /* "Nigeria"      */
    M1X22_CNTRY_CODE_NO           = 48,     /* "Norway"       */
    M1X22_CNTRY_CODE_OM           = 49,     /* "Oman"         */
    M1X22_CNTRY_CODE_PK           = 50,     /* "Pakistan"     */
    M1X22_CNTRY_CODE_PR           = 51,     /* "Peru"         */
    M1X22_CNTRY_CODE_PH           = 52,     /* "Philippines"  */
    M1X22_CNTRY_CODE_PL           = 53,     /* "Poland"       */
    M1X22_CNTRY_CODE_PT           = 54,     /* "Portugal"     */
    M1X22_CNTRY_CODE_RO           = 55,     /* "Romainia"     */
    M1X22_CNTRY_CODE_RU           = 56,     /* "Russia"       */
    M1X22_CNTRY_CODE_SA           = 57,     /* "Saudi Arabia" */
    M1X22_CNTRY_CODE_SG           = 58,     /* "Singapore"    */
    M1X22_CNTRY_CODE_SK           = 59,     /* "Slovakia"     */
    M1X22_CNTRY_CODE_SI           = 60,     /* "Slovenia"     */
    M1X22_CNTRY_CODE_ZA           = 61,     /* "S. Africa"    */
    M1X22_CNTRY_CODE_KR           = 62,     /* "S. Korea"     */
    M1X22_CNTRY_CODE_ES           = 63,     /* "Spain"        */
    M1X22_CNTRY_CODE_SE           = 64,     /* "Sweden"       */
    M1X22_CNTRY_CODE_CH           = 65,     /* "Switzerland"  */
    M1X22_CNTRY_CODE_SY           = 66,     /* "Syria"        */
    M1X22_CNTRY_CODE_TW           = 67,     /* "Taiwan"       */
    M1X22_CNTRY_CODE_TH           = 68,     /* "Thailand"     */
    M1X22_CNTRY_CODE_AE           = 69,     /* "UAE"          */
    M1X22_CNTRY_CODE_UK           = 70,     /* "UK"           */
    M1X22_CNTRY_CODE_US           = 71,     /* "USA"          */
    M1X22_CNTRY_CODE_YE           = 72,     /* "Yemen"        */
}
M1X22_COUNTRY_CODE;
```

4.4 M1X22_HOOK_SWITCH

Description

Hook switch command for on-hook and off-hook.

Prototype

```
/*
*****************************************************************************
** Hook switch
*****************************************************************************/
typedef enum {
    M1X22_ON_HOOK  = 0,
    M1X22_OFF_HOOK = 1,
}
M1X22_HOOK_SWITCH;
```

Name	Value	Description
M1X22_ON_HOOK	0x00	On hook.
M1X22_OFF_HOOK	0x01	Off hook.

4.5 M1X22_REG_TYPE

Description

List of 1x22 device register type.

Note: Register 0x00 and 0x11 are not available. Reading from or writing to these registers will not have any effect.

Prototype

```
/*
**************************************************************************
** M1x22 Register Type
**************************************************************************/
typedef enum {
    M1X22_REG_0x00 = 0x0,
    M1X22_REG_0x01 = 0x1,
    M1X22_REG_0x02 = 0x2,
    M1X22_REG_0x03 = 0x3,
    M1X22_REG_0x04 = 0x4,
    M1X22_REG_0x05 = 0x5,
    M1X22_REG_0x06 = 0x6,
    M1X22_REG_0x07 = 0x7,
    M1X22_REG_0x08 = 0x8,
    M1X22_REG_0x09 = 0x9,
    M1X22_REG_0x0A = 0x0a,
    M1X22_REG_0x0B = 0x0b,
    M1X22_REG_0x0C = 0x0c,
    M1X22_REG_0x0D = 0x0d,
    M1X22_REG_0x0E = 0x0e,
    M1X22_REG_0x0F = 0x0f,
    M1X22_REG_0x10 = 0x10,
    M1X22_REG_0x11 = 0x11,
    M1X22_REG_0x12 = 0x12,
    M1X22_REG_0x13 = 0x13,
    M1X22_REG_0x14 = 0x14,
    M1X22_REG_0x15 = 0x15,
    M1X22_REG_0x16 = 0x16,
    M1X22_REG_0x17 = 0x17,
    M1X22_REG_0x18 = 0x18,
    M1X22_REG_0x19 = 0x19,
    M1X22_REG_0x1A = 0x1a,
    M1X22_REG_0x1B = 0x1b,
    M1X22_REG_0x1C = 0x1c,
    M1X22_REG_0x1D = 0x1d,
    M1X22_REG_0x1E = 0x1e,
    M1X22_REG_0x1F = 0x1f
}
M1X22_REG_TYPE;
```

Parameters

Name	Value	Description
M1X22_REG_0x00	0x00	Register 0x00
M1X22_REG_0x01	0x01	Register 0x01
M1X22_REG_0x02	0x02	Register 0x02
M1X22_REG_0x03	0x03	Register 0x03
M1X22_REG_0x04	0x04	Register 0x04
M1X22_REG_0x05	0x05	Register 0x05
M1X22_REG_0x06	0x06	Register 0x06
M1X22_REG_0x07	0x07	Register 0x07
M1X22_REG_0x08	0x08	Register 0x08
M1X22_REG_0x09	0x09	Register 0x09
M1X22_REG_0x0A	0x0A	Register 0x0A
M1X22_REG_0x0B	0x0B	Register 0x0B
M1X22_REG_0x0C	0x0C	Register 0x0C
M1X22_REG_0x0D	0x0D	Register 0x0D
M1X22_REG_0x0E	0x0E	Register 0x0E
M1X22_REG_0x0F	0x0F	Register 0x0F
M1X22_REG_0x10	0x10	Register 0x10
M1X22_REG_0x11	0x11	Register 0x11
M1X22_REG_0x12	0x12	Register 0x12
M1X22_REG_0x13	0x13	Register 0x13
M1X22_REG_0x14	0x14	Register 0x14
M1X22_REG_0x15	0x15	Register 0x15
M1X22_REG_0x16	0x16	Register 0x16
M1X22_REG_0x17	0x17	Register 0x17
M1X22_REG_0x18	0x18	Register 0x18
M1X22_REG_0x19	0x19	Register 0x19
M1X22_REG_0x1A	0x1A	Register 0x1A
M1X22_REG_0x1B	0x1B	Register 0x1B
M1X22_REG_0x1C	0x1C	Register 0x1C
M1X22_REG_0x1D	0x1D	Register 0x1D
M1X22_REG_0x1E	0x1E	Register 0x1E
M1X22_REG_0x1F	0x1F	Register 0x1F

4.6 M1X22_DEBUG_TRACE_MASK

Description

Defines all debug trace mask.

Prototype

```
/*
**************************************************************************
** Debug Trace Mask
**************************************************************************/
typedef enum {
    M1X22_TRACE_MASK_EVENT      = M1X22_DEBUG_EVENT,
    M1X22_TRACE_MASK_INIT       = M1X22_DEBUG_INIT,
    M1X22_TRACE_MASK_PATH       = M1X22_DEBUG_RING_PATH,
    M1X22_TRACE_MASK_TRACE      = M1X22_DEBUG_TRACE,
    M1X22_TRACE_MASK_CNTRY_CODE = M1X22_DEBUG_COUNTRY_CODE,
    M1X22_TRACE_MASK_LINE_STATE = M1X22_DEBUG_LINE_STATE,
    M1X22_TRACE_MASK_IOCTL      = M1X22_DEBUG_IOCTL,
    M1X22_TRACE_MASK_BARRIER    = M1X22_DEBUG_BARRIER,
    M1X22_TRACE_MASK_INT        = M1X22_DEBUG_INT,
    M1X22_TRACE_MASK_PHU        = M1X22_DEBUG_PHU,
    M1X22_TRACE_MASK_KPROC      = M1X22_DEBUG_KPROC,
    M1X22_TRACE_MASK_ERROR      = M1X22_DEBUG_ERROR
}
M1X22_DEBUG_TRACE_MASK;
```

Parameters

Name	Value	Description
M1X22_TRACE_MASK_EVENT	0x00000001	Event trace mask.
M1X22_TRACE_MASK_INIT	0x00000002	Initialization trace mask.
M1X22_TRACE_MASK_RING_PATH	0x00000004	Ring path analysis trace mask.
M1X22_TRACE_MASK_TRACE	0x00000008	Function level trace mask.
M1X22_TRACE_MASK_CNTRY_CODE	0x00000010	Country code processing trace mask.
M1X22_TRACE_MASK_LINE_STATE	0x00000040	Line state analysis path trace mask.
M1X22_TRACE_MASK_IOCTL	0x00000080	IOCTL trace mask.
M1X22_TRACE_MASK_BARRIER	0x00000200	Barrier related function trace mask.
M1X22_TRACE_MASK_INT	0x00000400	Interrupt trace mask.
M1X22_TRACE_MASK_PHU	0x00000800	Phone hung up in off-hook trace mask.
M1X22_TRACE_MASK_KPROC	0x00002000	Kernel processing trace mask.
M1X22_TRACE_MASK_ERROR	0x80000000	Error trace mask.

5 Sample Application

This section illustrates how an application can be implemented using the Modem APIs. The sample program is a typical telephony application that automatically answers the call when a ring burst is detected. This is accomplished by simply going off-hook.

Note: This sample program will definitely not compile and is not a complete application. It is intended only for demonstration purpose to show the capability and to illustrate the interaction between the application program and the Modem API.

```
/*************************************************************
**
**   sample application
**
**   Perform the following:
**   - Initialize the API subsystem – M1X22_ApiInit()
**   - Open the device               - M1X22_OpenDevice()
**   - Open each channel            - M1X22_OpenChannel()
**   - Initialize channel           - M1X22_InitChannel()
**
*************************************************************/
int main (int argc, char **argv)
{
      M1X22_CHAN_INIT   chanInit;
      M1X22_HANDLE      pFxoDev;

      /* Initialize the API subsystem */
      if (M1X22_MdmApiInit() != M1X22_SUCCESS)
            return (-1);

      /* Open device descriptor with event callback function */
      pFxoDev = M1X22_OpenDevice("/dev/ter", &fxoEventCallback);
      if (pFxoDev == NULL)
            return (-1);

      /* For each channel descriptor */
      for (i=0; i<MAX_DAA_INSTALLED; i++)
      {
            /* Open the channel descriptor */
            channles[i].pHandle = M1X22_OpenChannel("/dev/ter", pFxoDev, i);
            if (channels[i].pHandle == NULL)
                  return (-1);

            /* Initialize the channel for operation in US */
            if (M1X22_InitChannel (channels[i].pHandle, M1X22_CNTRY_CODE_US)
            != M1X22_SUCCESS)
                  return (-1);
      }

      /* program main loop */

      While (1)
      {
            /* this is the main program loop…
            /* . . . */
      }

      /* Release the API subsystem before exiting */
```

```
      M1X22_RET M1X22_MdmApiRelease (void);

      Return (0);
}

/*
**************************************************************************
**********************
* FUNCTION NAME:
*   fxoEventCallback
*
* DESCRIPTION:
*   This is the event callback function. It is invoked by the API
*   subsystem when an event occurs. Currently, this callback function
*   only handles the RING start event. It takes the line off-hook when
*   the ring burst is detected.
*
* PARAMETERS:
*   event_id     - Event ID
*   channelId    - channel where event is occurring
*   data1        - Additional data (1)
*   data2        - Additional data (2)
*   data3        - Additional data (3)
*   data4        - Additional data (4)
*
* RETURNS:
*   N/A
**************************************************************************
**********************
*/
void fxoEventCallback (M1X22_EVENT_ID event_id,
                       unsigned int channelId,
                       unsigned int data1,
                       unsigned int data2,
                       unsigned int data3,
                       unsigned int data4)
{
      int ret;
      M1X22_HANDLE  pHandle;

      pHandle = channels[channelId].pHandle;

      switch (event_id)
      {
            case M1X22_EVENT_RING_START:
                  printf("\nRING(%d)",channelId);
                  printf("\n     Frequency: %d(hz)",data1);
                  printf("...auto answer...\n");

                  ret = M1X22_HookSwitch (pHandle, M1X22_OFF_HOOK);
                  if (ret == M1X22_FAILED)
                        break;
                  else
                        printf("\nSuccessful");
                  break;

        case M1X22_EVENT_RING_END:
             printf("\nRING_END(%d)\n\r",channelId);
```

```
            printf("\n     Frequency: %d(hz), Duration:
%d(ms)\n",data1,data2);
            break;

        case M1X22_EVENT_POLARITY_REV:
            printf("\nPOLARITY_CHANGE(%d)\n\r",channelId);
            break;

        case M1X22_EVENT_BATT_FEED:
            printf("\nBATTERY_FEEDED(%d)\n\r",channelId);
            break;

        case M1X22_EVENT_BATT_DROP:
            printf("\nBATTERY_DROPPED(%d)\n\r",channelId);
            break;

        case M1X22_EVENT_APOH:
            printf("\nAPOH(%d)\n\r",channelId);
            break;

        case M1X22_EVENT_NOPOH:
            printf("\nNOPOH(%d)\n\r",channelId);
            break;

        case M1X22_EVENT_LINE_STATE:
            printf("\n\r   LINE_STATE(%d)",channelId);
            printf("\n     HookState: %s, Entity: %s, Row: %d, Event: %d\n",

M1X22_GetHookStateStr(data1),M1X22_GetEntityStr(data2),data3,data4);
break;

        default:
            printf("\nUnkown event\n\r");
            break;
    }
}
```

6 Related Documentation

The following 73M1x22 documents are available from Teridian Semiconductor Corporation:

73M1822/73M1922 Data Sheet
73M1822/73M1922 Layout Guidelines
73M1x22 Worldwide Design Guide
73M1822/73M1922 Control Module User Guide
73M1822/73M1922 Hardware Module for SMDK412 User Guide
73M1822/73M1922 Modem API User Guide
73M1822/73M1922 Modem CTL Application User Guide
73M1822/73M1922 MicroDAA Software Architecture

7 Contact Information

For more information about Teridian Semiconductor products or to check the availability of the 73M1822 and 73M1922, contact us at:

6440 Oak Canyon Road
Suite 100
Irvine, CA 92618-5201

Telephone: (714) 508-8800
FAX: (714) 508-8878
Email: modem.support@teridian.com

For a complete list of worldwide sales offices, go to http://www.teridian.com.

———

Revision History

Revision	Date	Description
1.0	12/23/2009	First publication.